

Article

Enabling End-Users in Designing and Executing of Complex, Collaborative Robotic Processes

Helmut Zörrer ¹ , Georg Weichhart ¹ , Mathias Schmoigl Tonis ^{2,*}, Till Bieg ³ , Matthias Propst ¹ , Dominik Schuster ⁴ , Nadine Sturm ⁵, Chloé Nativel ⁶, Gabriele Salomon ⁵, Felix Strohmeier ² , Andreas Sackl ³ , Michael Eberle ⁴ and Andreas Pichler ¹ 

- ¹ PROFACOR GmbH, 4407 Steyr-Gleink, Austria; helmut.zoerr@profactor.at (H.Z.); georg.weichhart@profactor.at (G.W.); matthias.propst@profactor.at (M.P.); andreas.pichler@profactor.at (A.P.)
 - ² Salzburg Research GmbH, 5020 Salzburg, Austria; felix.strohmeier@salzburgresearch.at
 - ³ Center for Technology Experience, Austrian Institute of Technology GmbH, 1210 Vienna, Austria; till.bieg@ait.ac.at (T.B.); andreas.sackl@ait.ac.at (A.S.)
 - ⁴ Eberle Automatische Systeme GmbH & Co KG, 6850 Dornbirn, Austria; dominik.schuster@eberle.at (D.S.); michael.eberle@eberle.at (M.E.)
 - ⁵ Johanniter Österreich Ausbildung und Forschung gem. GmbH, 1210 Vienna, Austria; nadine.sturm@johanniter.at (N.S.); gabriele.salomon@johanniter.at (G.S.)
 - ⁶ Regionales Innovations Centrum GmbH, 4623 Gunskirchen, Austria; office@ric.at
- * Correspondence: mathias.schmoigl-tonis@salzburgresearch.at

Abstract: Over the last years, capabilities of robotic systems have quantitatively and qualitatively improved. But going beyond isolated robotic systems, the integration and interoperability of robotic capabilities in complex work processes remains a major challenge. This lack of tools to integrate robots needs to be addressed on technical, semantic and organizational level. In the ROBxTASK research project, we developed an approach to support cooperation between different types of users in order to enable domain experts, with no robotic know-how, to work with robot-assisted workflows. By engineering robotic skills at a useful and usable level of abstraction for experts in different domains, we aim to increase re-usability of these skills on two different levels, (robotic) device level, and on level of application specific workflows. The researched prototype consists of a web platform, which allows (a) engineers to register (robotic) devices and the implemented skills of the devices, (b) domain experts to use a graphical task design environment to create workflows across multiple robotic devices and lastly (c) robot co-workers to download and execute the workflow code in a local environment with digital twins or real robots. Additionally skills and workflows can be shared across organisations. Initial user studies have shown that the visual programming environment is accessible and the defined skill-set is easy to understand even for domain experts that are inexperienced in the field of robotics.

Keywords: visual programming environment; skill-based programming; digital twin; robotic devices; end-user programming; healthcare robotics



Citation: Zörrer, H.; Weichhart, G.; Schmoigl Tonis, M.; Bieg, T.; Propst, M.; Schuster, D.; Sturm, N.; Nativel, C.; Salomon, G.; Strohmeier, F.; et al. Enabling End-Users in Designing and Executing of Complex, Collaborative Robotic Processes. *Appl. Syst. Innov.* **2023**, *6*, 56. <https://doi.org/10.3390/asi6030056>

Academic Editors: José Gonçalves, Paulo Costa and Vítor H. Pinto

Received: 24 March 2023

Revised: 18 April 2023

Accepted: 9 May 2023

Published: 12 May 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Robot technologies are already used in very varying areas of our society. According to the International Federation of Robotics (IFR) [1] the World Robotics 2021 Industrial Robots report shows a record of 3 million industrial robots operating in factories around the world—an increase of 10% compared to 2020. The capabilities of robots have increased, with a high velocity where, for example between 1995 and 2016, the number of robot-related patent families worldwide have increased exponentially [2].

Within the service sector the market for Autonomous Mobile Robots (AMRs) is growing fast [3]. According to a recent market research study [4] containing the analysis of more than 500 players and 15+ geographical countries/regions, Automated Guided Vehicle

(AGV) and AMR Market is expected to reach \$13.2 Billion by 2026 with a growth rate of around ~35%. “Mobile robots and AGVs are used in industrial automation applications and as service robots in new environments such as hospitals.” [5] (p. 2). Other application areas for mobile robots are production logistics, agriculture, and service [6].

New domains for the application of social robots [7] have emerged. Among others, in health and daily care services the potentials of social robots have been demonstrated [8].

Implementing robotic solutions in a domain requires the expertise and collaboration of human resources from many disciplines. This is in particular true in complex environments like manufacturing [9]. Here, the missing workforce requires an increased level of flexible automation [10].

Approaches that promise less manual effort, such as data-driven approaches [11,12], are likely to fail due to missing data and changing processes. A model-driven approach is desired, requiring expert knowledge of both the robotic systems and the application domain in which the robotic processes are executed. In the industrial environment, holistic robotics solutions are often easier to implement due to the proximity of technical tasks that usually dominate there. In domains like healthcare, involving domain experts can be more difficult, because in many cases they do not have any robotic know-how. In any case, knowledge of a complex and unique situation is needed, where requirements and processes of the application domain, in general, and in the particular of the to-be automated use-case, is combined with the technical knowledge how to program collaborative robots interacting with humans and the other system along the processes. Interoperability needs to be established on technical, semantic and organizational process level [13,14].

Training people from the application domain in computational thinking for being able to program robotic processes requires effort [15]. The same is true in training robot programmers in understanding the particularities of the end-user domain. What is needed, is an approach that allows people either with robotic know-how to easily understand the requirements and the processes of the application domain, or an approach where (application) domain experts can easily model robotic processes.

In the project ROBxTASK, we have focused on the second approach [16–19]. In brief, in this project we support multiple user groups working together on the task of creating robotic programs. The three roles supported are: (a) the Robotic Engineer to provide concrete implementations of robotic skills for a specific hardware; (b) the Task Designer, a person with partial domain and robotic know-how, is working on robotic workflows that build on the skills provided by (a); (c) the Robot Co-Worker is an end-user domain expert with no robotic knowledge, but who is working with robots and has the responsibility to execute the workflows designed by (b).

Application-oriented research of accessible interfaces that allow end-users to build robot-based applications has already been researched [20,21] including applications for robot-based therapy [22–24], education [25], social robotics [26,27], industry [28] and service robots [29]. A broad overview on the topic of end user robotic programming is presented by Ajaykumar, Steele, and Huang [30]. In general we found the following platforms and research of particular interest:

- **Trigger-Action Programming:** The trigger-action paradigm [31], is a possible approach to help application domain experts, without specific programming know-how, to personalize the behaviour of humanoid robots. The tool IFTTT (<https://ifttt.com>, accessed on 8 May 2023) is a well known example, implementing this paradigm.
- **Code3:** A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts [32]. Code3 builds on Google’s TMBlockly (<https://developers.google.com/blockly>, accessed on 8 May 2023) [33] environment. However there the focus is on single robots. The focus with respect to targeted user group is on programmers (with and without robotic coding skills). Code3 enables use-case-specific code using three parts “(1) CustomLandmarks for developing perceptual capabilities, (2) CustomActions for programming manipulation actions, and (3) CodeIt for combining the two in a high-level program that captures task requirements” [32] (p. 2).

Use-cases and robot specific Landmarks and Actions are defined in Code3 and can then be used in the CodeIt environment. This environment shows a lower level of abstraction, which gets visible by code that includes detailed modelling of gripper movements [32].

- CoSTAR [34] a system that allows to use natural abstractions and perception in a way users can both understand and utilize to author plans using behavior trees and high-level information from perception of known objects. A certain skill for behaviour trees is needed. Also the behaviour elements needs to be programmed in a specific way for the targeted systems.
- RAZER [35] is a software framework for task-level programming. The system targets both user groups, robotic experts and shop-floor workers. The first user group is supported by a web-based graphical interface providing robotic skills and their interface. The workers are supported by the graphical interface in creating tasks from parameterized skills. Although the system has been used to program various hardware, the focus is on robots.
- ABB Wizard easy programming (<https://new.abb.com/products/robotics/application-software/wizard>, accessed on 8 May 2023) and ABB Robot Studio (<https://new.abb.com/products/robotics/de/robotstudio>, accessed on 8 May 2023) [36] are exemplary approaches provided by a company for their proprietary robotic systems. Wizard is the standard graphical programming interface for ABB cobots and uses a drag-and-drop paradigm as opposed to coding. The second system, the robot-studio allows off-line creation and programming of virtual twins of a robot.
- LabVIEW [37] is graphical software for test, measurement, control and regulation applications. In the foreground are system developments and integration tasks that require quick access to the hardware and a detailed insight into the recorded data (<https://www.ni.com/de-at/shop/software/products/labview.html>, accessed on 8 May 2023). LabVIEW can be enhanced with robotic modules e.g., for LEGOTMMINDSTORMSTMRobot Inventor (<https://www.lego.com/en-at/product/robot-inventor-51515>, accessed on 8 May 2023) (which can be used stand-alone). As such the targeted user group are expert engineers for LabVIEW. For the MINDSTORMS system the (obvious) user group are kids interested in robotics.

While these applications are highly valuable for making robot programming more accessible to end-users, they are partially constrained to certain use cases, application domains, commercial technology providers or specific hardware. In contrast to domain-specific approaches, interfaces for end-user-friendly robot programming that are applicable across different domains and hardware would bring remarkable benefits, since robotic tasks in different application domains are usually similar. For example, picking up an item and placing the item in another location is a task that is common in both industrial and healthcare domains [18]. Thus, applications that enable both the easy development of robot-based tasks and the exchange of tasks between different application domains bear great potential, as they can support transfer of knowledge between different domains, decrease redundancies, and ultimately reduce time and costs.

The contributions of our research can be summarized as follows:

- Development of a scalable micro-service based open source web platform ROBxTASK (<https://github.com/ROBxTASK>, accessed on 8 May 2023), to support cooperation between user groups, enable domain experts to develop robotic workflows using the Visual Programming Language (VPL) Google Blockly and generate executable code.
- Identification of appropriate skills at the right level of abstraction for domain experts inexperienced in robotics for cross-domain and cross-device usage for the healthcare and industrial domain.
- Improving test ability of distributed workflows by enabling testing without robotic hardware and supporting integration of a digital twin.
- Evaluation of the operational capability of the platform in healthcare and industry use cases.

- User-centered evaluation of the platform for task designers based on three user studies to test the usability of the developed platform for the selected target group.

The paper is organized as follows. Section 2 gives an overview of the ROBxTASK platform. In Section 3 the use cases that were used in the project to evaluate the platform are explained. The user studies evaluated in the research project are then described in detail in Section 4. Section 5 discusses the methods and results. Finally, our paper is concluded in Section 6.

2. ROBxTASK Overview

In order to introduce the reader to the general context of this work, we describe user roles, followed by the interaction between roles.

As briefly mentioned in Section 1, the collaboration platform provides an infrastructure that connects three types of users. This allows to build an organisational workflow to link detailed and hardware-specific implementations with domain experts and robotic end-users.

The three user roles are described in more detail in Figure 1 and in the following.

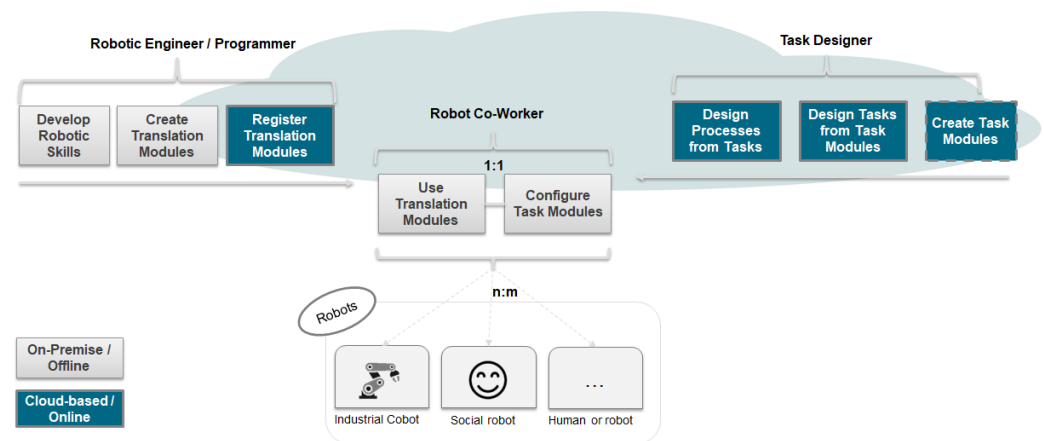


Figure 1. ROBxTASK user roles.

- **Robotic Engineer:** Users in this role, do not have and do not need any end-user specific know-how. They focus on implementing skills for a specific hardware.
- **Robot Co-Worker:** Users in this role are focusing on the execution of processes. The robot is a tool only. Users in this role have no specific robotic know-how. For this type of user, the focus is on efficiency and effectiveness of execution.
- **Task Designer:** Users in this role bridge the gap between end-user domain-specific needs for executions of processes on the one side and robot-specific implementations of tasks that are needed within the processes. Task Designers need know-how in robotic and end-user domain. However, ROBxTASK support eliminates the need for detailed knowledge for users in this role. The tools and approach supports abstraction of details from both domains. Organizational and tooling support includes reuse and sharing of both process models and task module implementations. The task designer is a system integrator on technical level and on organisational level. The platform supports users in this role to collaborate with both other roles. Robotic engineers are contacted to create task-implementations [38] for robotic systems and on the other side robot co-workers need workflows to execute their processes (Figure 2).

The ROBxTASK platform uses Google's TMBlockly (<https://developers.google.com/blockly>, accessed on 8 May 2023)-Editor on the frontend, which allows users to build appropriate actors/software agents [39–41] by using the skills of the corresponding robotic devices.

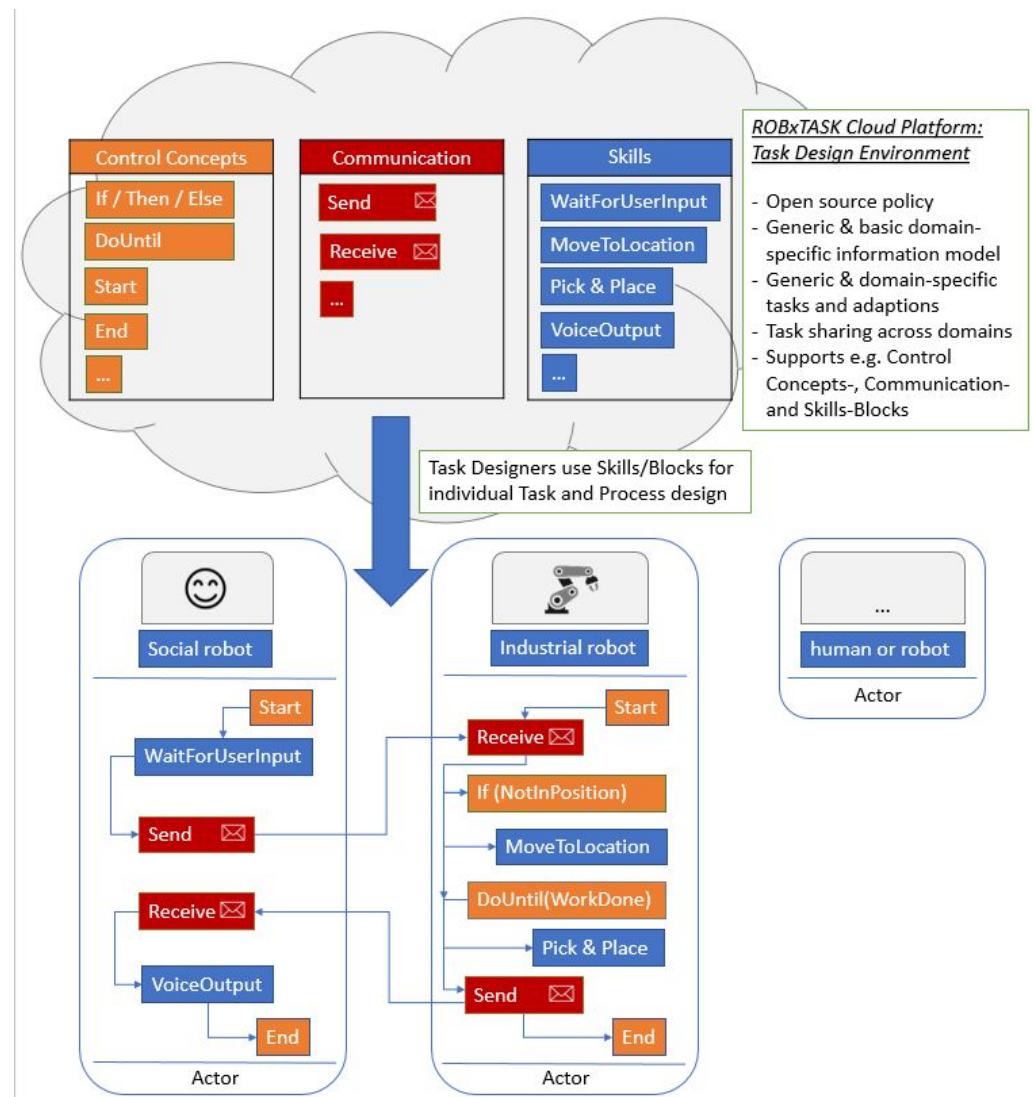


Figure 2. ROBxTASK Task and Process Design and Exchange Environment.

The full workflow to create and run an application involves all previously described user roles and can be simplified into five steps:

1. **Robotic Engineer:** Implement skills and register a robot on the platform. An engineer needs to implement a server application (e.g., ROS action Server or OPC UA Server), create a JavaScript Object Notation (JSON) based registration file containing all available skills of the target robot, and register the device on the platform.
2. **Task Designer:** Select a robot target platform for task programming.
3. **Task Designer:** Create a task design in the Blockly task design editor by using the available robotic skills and combining task modules in any way. May also design a distributed process using multiple robots/robotic devices (see Figure 3).
4. **Task Designer:** Save the task design project.
5. **Robot Co-Worker:** Start the project download and run the generated program agents/actors locally. Note that all robots should be in the same intranet for shared execution.

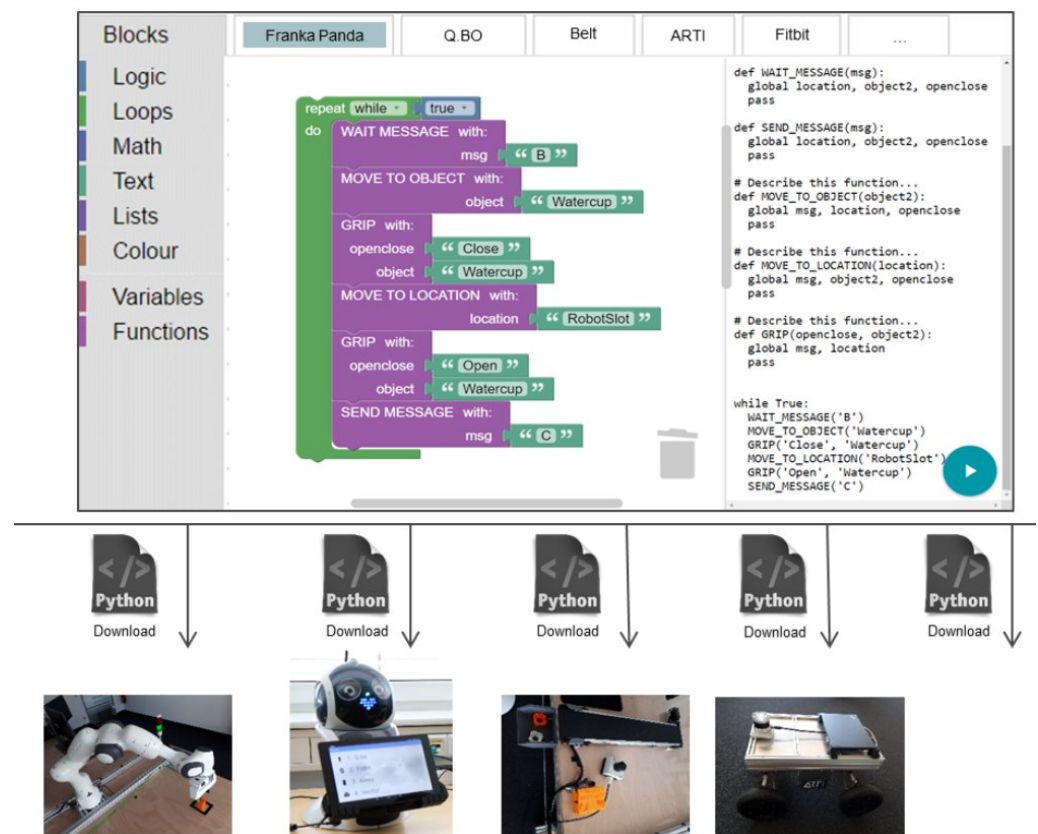


Figure 3. Symbolic overview of handling multiple robots/robotic devices in ROBxTASK.

2.1. Skills in ROBxTASK

Skill-based programming simplifies the creation of robot programs, and helps to enable non-experts to create robot applications [42]. Thus, a lot of research with respect to skill-based programming can be found in literature especially for the industrial area [43–46]. Following Pedersen et al. [47] approaches of task-level programming using skills can be divided into three layers:

- Primitives: Perform a single operation related to the robot system.
- Skills: Are combination of primitives.
- Tasks: Are directly related to solving goals and can be either explicitly programmed or planned ad hoc, if skills are e.g., conform to the STRIPS [48] notation.

Some approaches also take care of supporting to find the right skill. Bøgh et al. [49] developed first a method to find re-usable skills through analysis of industrial tasks, industrial implementations and laboratory experiments; secondly they analyse Standard Operation Procedures (SOP).

The result of the analysis finds that only a few standard skills are needed e.g., to support flexible mobile manipulators for industrial logistics and assistive scenarios: Move To, Locate, Pick up, Place, Unload, Shovel, Check, Align, Open, Close, Press, Release and Turn.

According to [42] (p. 1) “the Skill Based Programming eases the robot program generation, its similarity to human behavior allows non expert operators maintaining, adapting or creating robotic applications”, so this approach was also chosen for implementing a visual programming environment in ROBxTASK.

The initial set of skills was drafted in a series of workshops with stakeholders from healthcare and industry domains. One of the major criterion for developing the initial skill was that a specific skill is needed for the description and later implementation of a use case (see Section 3) from the perspective of the task designer: In the workshops the MIRO boards (<https://miro.com>, accessed on 8 May 2023) (see Figure 4) were used to identify the skills,

skill slot parameters and skill slot types presented in this chapter. The MIRO software has proven to be extremely useful in picking up the different groups of participants in the research project with their different levels of knowledge at an appropriate level.

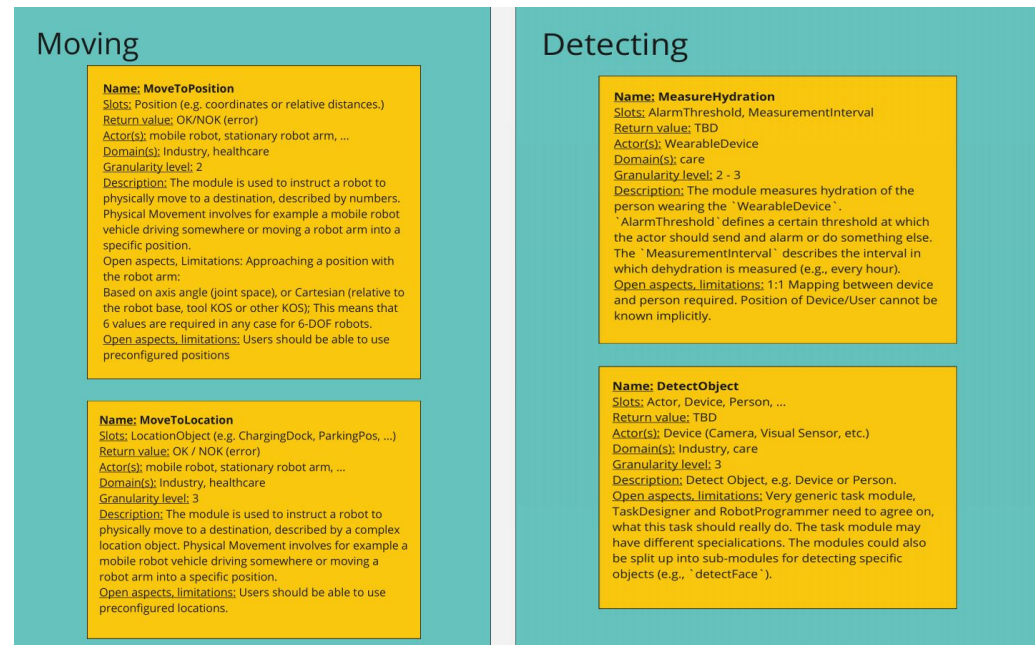


Figure 4. Planning skills and tasks in MIRO boards.

Multiple evaluation studies have been conducted to find the right level of abstraction regarding specifically the skill formal definition, slot parameters, return values and applied skill semantics. The slot types that can be used in module descriptions have been initially defined in the workshops of the project. However, during progress of work we have been reducing the slot types to:

- “String”
- “Numeric”
- “Enum” (e.g., list of predefined position from which the user can choose)
- “Bool”

To enable heterogeneous robots to agree on a basis of skills that can be programmed in the editor, the project developed a skillset that can be targeted by concrete robots while registration of a new robotic system on the platform. The target version of the skill set implementation of the robot has to be provided upon registration. Table 1 shows our defined skills (ROBxTASK skillset version 1.0) and all blocks that are available in the task design editor for programming agents/actors.

Concrete robots can communicate their provided subset of the skillset using a JSON-based registration file (details about registration process following in Section 2.2).

Similar to the servicelevel used in [50] we can add other runtime environment (RTE) features later to provide additional blocks for task programming. This architecture enables us to support building e.g., more intelligent software agents later, by providing AI features, database access, ... if needed.

Table 1. Blocks implemented in the ROBxTASK Blockly environment (☑ RTE Feature, ⚡ Robotic Skill, ≡ Programming Feature).

Group	Skill/Block
Communication (Basic Interoperability)	☑ SendMessage ☑ OnMessageReceive
Moving	⚡ MoveToLocation ⚡ DeliverObject ⚡ FetchObject
Detecting	⚡ MeasureHydration ⚡ DetectObject
Grabbing and Unloading	⚡ GrabObject ⚡ PutObject
Setup and Teaching	⚡ TeachingWorkspacePosition ⚡ TeachingObjectRecognition
Basic Control Concepts	≡ Loop ≡ Selection
External Interaction	⚡ WaitForCondition ⚡ WaitForExternalEvent ⚡ WaitForUserInput ⚡ VoiceOutput ⚡ GraphicalUserInteraction
Setting and Getting Data of External Systems	⚡ GetData ⚡ SetData

2.2. ROBxTASK Platform

The ROBxTASK cloud platform is based on the microservice architecture of the “collaboration network for industry, manufacturing, business and logistics in Europe platform” (NIMBLE) [51]. NIMBLE (<https://www.salzburgresearch.at/projekt/nimble>, accessed on 8 May 2023) is a cloud-based, Industrie 4.0, Internet-of-things-enabled B2B platform on which European manufacturing firms can register, publish machine-readable catalogues for products and services, search for suitable supply chain partners, negotiate contracts and supply logistics, and develop private and secure B2B and M2M information exchange channels to optimise business workflows. Using the existing technology stack from NIMBLE, the ROBxTASK platform can support features like, e.g., user management, registration of items on the platform, searching for registered items or chatting with other users. Some sample dialogs of the platform can be found in Appendix A.

Figure 5 shows the Task Design editor, where all registered device skills are available as blocks to be used in Google Blockly. In the Task Design editor, the total workflow can be saved and restored in XML. Finally, the code for a concrete RTE (e.g., “ROS”) can be downloaded as a zip file which contains for any device used in the workflow the concrete python agent code. Thus, these agent files can be used by the robot co-worker then to execute the workflow.

When the Robot Co-Worker presses the button “</> OPCUA”, the flexible generator (<https://github.com/ROBxTASK/codegen-service>, accessed on 8 May 2023) generates Python code that supports our ROBxTASK RTE (RxRTE) developed within the project [19]. RxRTE also supports the use of different test environments during the execution of the workflow, e.g., to test workflows without hardware by using mock objects (skill calls are then only emulated by waiting, e.g., 2 s) or to simulate the workflow using a digital twin, and enables the monitoring of the distributed workflow using some additional tools (see Section 3.2.1).

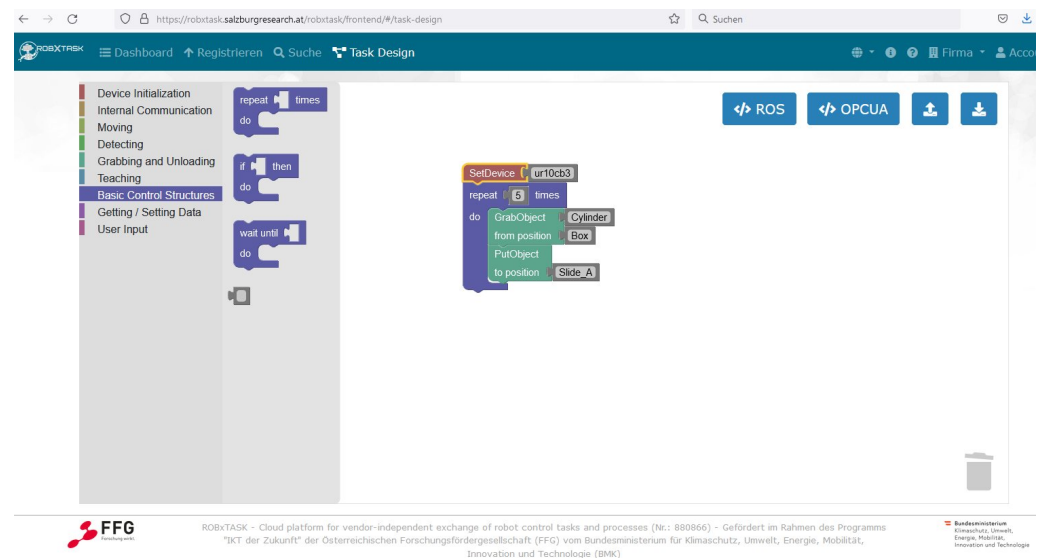


Figure 5. Sample of the ROBxTASK Task Design Editor.

The ability to support digital twins proved to be helpful for task designers too. In a later stage of the project we developed a docker based testing environment where task designers can now download their programs from the platform and test them using digital twins which support our use cases. This new environment simplifies the understanding of the programs for task designers, and is described in more detail in Section 4.3.

3. Use Cases

A central goal of the ROBxTASK project is to demonstrate and validate the functionality of the platform based on practice-oriented use cases. To promote the cross-domain usability of the platform, use cases in healthcare and industry have been planned.

To identify meaningful use cases for both application domains, design fiction workshops with stakeholders from industry and healthcare domains were conducted as part of the human-centered development process in ROBxTASK. In the workshops, stakeholders were asked to brainstorm useful robot-based applications for the respective domains. Next, these ideas were further developed into concrete use cases, also taking their practical feasibility into account. The different use cases were further refined by members of the project consortium as described in Sections 3.1 and 3.2. To test the cross-device use of our skills, we planned to use different robotic devices in the use cases for the same robotic tasks, e.g., for grasping/lying operations (skills: GrabObject, PutObject) and transport tasks (skill: MoveToLocation).

3.1. Use Case Healthcare (UCH)

In contrast to the manufacturing industry the Nursing and Care industry is dominated by direct human interaction in a physical and social manner. That fact rises specific considerations about the implementation of robotic systems within this environment. Societal and ethical issues like autonomy, dignity, acceptance and responsibility are the primary subjects of the debate nevertheless, in view of the increasingly aging society and simultaneous shortage of skilled workers within the nursing and care sector, assisting robotic systems can be a building block to relieve that tensioned situation [52].

Different activities on social and service robots are already on the rise. Some prominent examples to name are the therapeutic robot seal PARO (<http://www.parorobots.com>, accessed on 8 May 2023) [53] or service robots like the Care-o-Bot (<https://www.care-o-bot.de/de/care-o-bot-4.html>, accessed on 8 May 2023) [54] from the Fraunhofer-Institute in Germany or LIO (<https://www.fp-robotics.com/de/lio>, accessed on 8 May 2023) [55] from F&P Personal Robotics.

The knowledge of predictable physical activities is crucial to identify further application fields. Estimates assume the proportion of such activities within nursing and care industry around 30–36% opposed to the manufacturing industry, where the amount of predictable activities is up to 59% [56].

In order to identify an applicable use case for nursing and healthcare scenarios in ROBxTASK, an analysis on the basis of “nursing phenomena” [57] in combination with a fictitious “nursing process” [58] was conducted. Therefore, two possible scenarios were selected, namely “malnutrition” and “high risk of falls”. Both variants were presented and discussed in the consortium, finally a consensus was reached on the malnutrition scenario. This is mainly explained by the consideration of possible technical implementation and practical added value for nursing professions. The process design was guided by the view of robotic systems as assistive devices in the day-to-day nursing process.

The flowchart in Figure 6 illustrates the whole malnutrition process. This process is characterized by the fact that intersections between health professions (nurses), task designers and the robotic system become visible. The process starts on the task designer side with the decision for whom the use case should be active and which assessment or screening tool should be applied. The robotic system independently performs the selected assessment or screening tool at predefined intervals and compares the data to predefined thresholds. If no deviation is detected, the robotic system performs a re-assessment at a specified interval. In the event of a detected deviation, an action message is sent by the robotic system to the health professionals. In addition, predefined measures take place with regard to close surveillance.

During the project implementation the use case was analyzed and discussed in detail. The joint decision was made from a feasibility perspective and represents a sub-process in the main malnutrition process, namely receive the request to bring beverages and monitor the fluid intake.

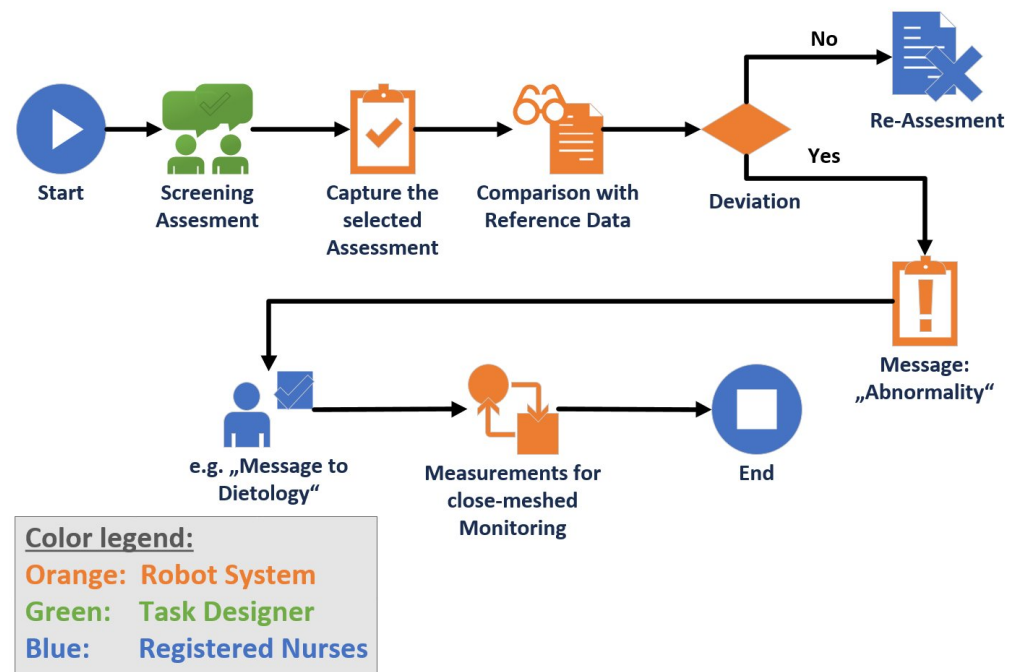


Figure 6. Flowchart of the malnutrition process.

The next step was to examine the use cases of industry and healthcare for synergies. The goal was to determine which robotic motion sequence can be similarly or even identically mapped. As a result, elements of the initial process could be used for further processing in the project.

In the healthcare domain we want to show that we can use the presented visual programming approach to enable an immobile patient to get drinks and food delivered to their patient bed. Since we want to show robot-cross execution we separated this delivery task into three different robots, each equipped with different individual unique skills. No single robot is able to perform the task alone, the robot always needs the skills of the others as well. Their shared task execution should work as described by the task designer in Blockly task design editor and collectively deliver an order to the patient bed.

For the healthcare domain we implemented a ROS-based local environment and registered three robots (see Figure 7) to the platform via JSON upload:

- Q.Bo One (<https://thecorpora.com>, accessed on 8 May 2023) (social robot by TheCorpora). Implements the following skills: GetData, SetData, GraphicalUserInteraction, VoiceOutput, WaitForUserInput, WaitForExternalEvent.
- ARTI Chasi robot (<https://arti-robots.com/project/product-chasi>, accessed on 8 May 2023) (mobile robot by ARTI). Implements the following skills: GetData, SetData, MoveToLocation, GraphicalUserInteraction, WaitForUserInput, WaitForExternalEvent.
- Franka Panda (<https://www.franka.de>, accessed on 8 May 2023) (arm robot by Franka Emika). Implements the following skills: GetData, SetData, WaitForUserInput, WaitForExternalEvent, GrabObject, PutObject.



Figure 7. Use case healthcare-Robots: TheCorpora Q.Bo One, ARTI Chasi and Franka Panda.

In the registration of the three robots the task designer can get additional information on how the robots are registered and which skills they do implement (decided and implemented by the robotic engineer). For the healthcare demonstration lab setup, we registered and implemented skills based on the minimum requirements needed to perform the scenario presented. After successful registration of these three robots, the task designer may for example design the workflow of the robots in the Blockly task design editor as follows (see also Figure 8):

1. Patients can interact with the social robot ‘Q.Bo One’ and place an order via audio input (e.g., say ‘Hugo, I need a cup of water!’). In our lab setup, there are nine different storage positions to choose food and drinks from, so the designer needs to hard-assign storage positions to audio commands.
2. Now the mobile robot ‘Chasi’ will autonomously drive to the drink station.
3. Next, the arm robot ‘Franka Panda’ will grasp the desired item and places the item on the mobile robot. The designer here needs to make sure that the mobile robot is always correctly parked in the parking slot first.
4. Then the mobile robot ‘Chasi’ will autonomously drive to the patient bed. Finally, after the item has been picked up, the robot should drive back to the original parking position.

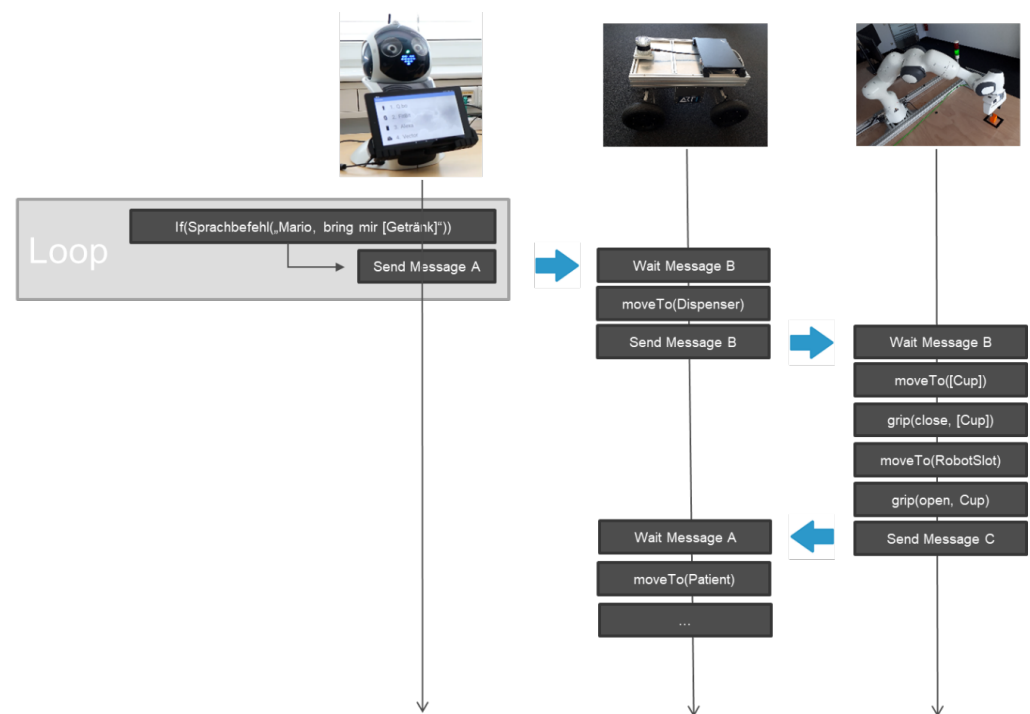


Figure 8. Use case healthcare-Initial planning of the workflow for ‘get a drink’.

In the end, how they want to implement the specific scenario is left up to the task designers. The goal however is to deliver an ordered item to an immobile patient lying in a bed and communicating with Q.Bo robot. Figure 9 shows an exemplary task design for the healthcare use case.

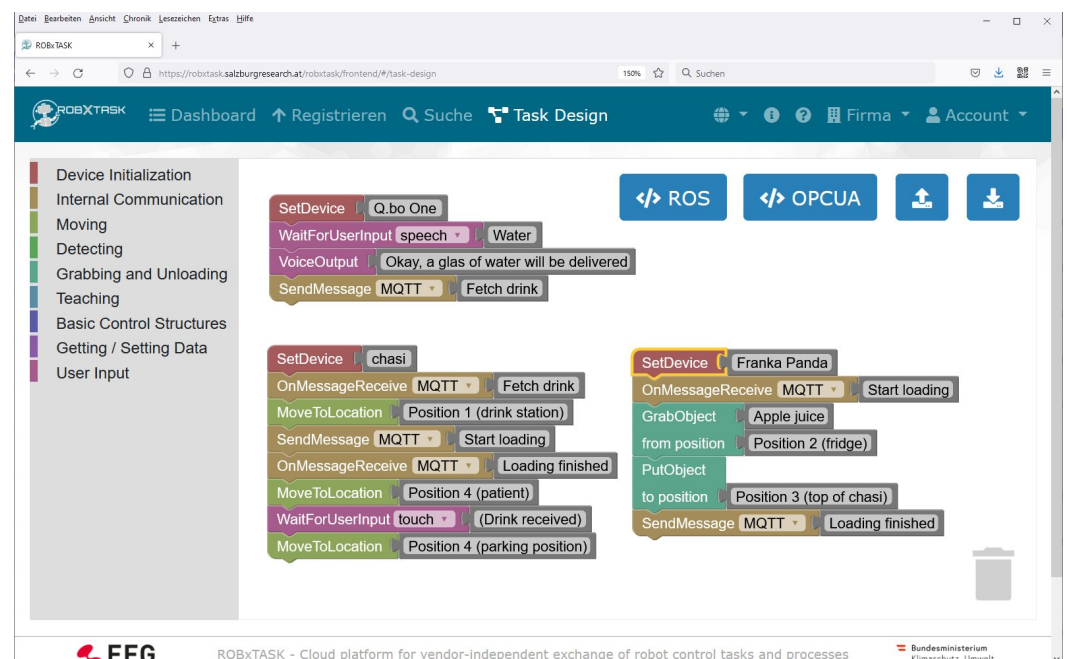


Figure 9. Sample Blockly code for the healthcare use case.

Developing the Healthcare Use Case by Using ROS

In the local environment when working with ROS we use the ROS visualization tool RViz (<http://wiki.ros.org/rviz>, accessed on 8 May 2023) and MoveIt (<https://moveit.ros.org>, accessed on 8 May 2023) for visualization and testing the execution of robotic skills triggered by the ROBxTASK system. The illustration in Figure 10 shows, how they are

connected in case of Franka Panda arm robot. This setting is used in the Salzburg Research Forschungsgesellschaft (SRFG) lab to showcase the health use case scenario.

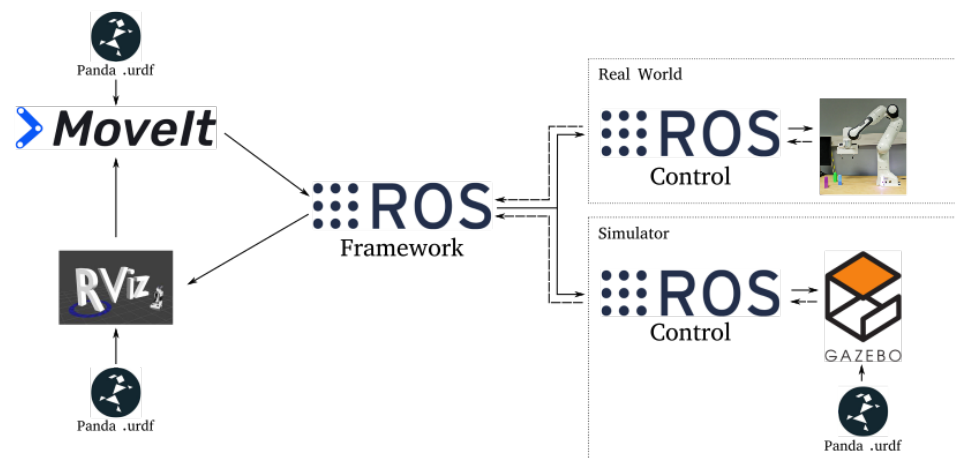


Figure 10. Connection of the ROS framework with MoveIt and RViz.

The ROBxTASK lab translation modules take commands from the ROS action clients (Python client) and communicate them to the robot's action server implementation (Python server). Each robot hosts his own action server and waits for ROBxTASK commands to execute given tasks. These action servers perform the action on the robot accordingly. Commands are communicated via intranet in Python-based ROS action topics with strictly defined message formats. Command sending success is evaluated using the Wireshark tool (<https://www.wireshark.org>, accessed on 8 May 2023). The ROS action clients are automatically generated by the codegenerator based on the compiled Task Design Editor project.

While the robots execute the command the planned paths of every movement are visualized for testing purposes in RViz before the actual movement is executed. This also allows stopping the robot before executing the movement by means of emergency stop functions and prevents self-damage of the robot due to bad ROBxTASK command designs. Since commands are derived from the graphical user interface Blockly application they will be accessible to non-technical users, which requires the robots to have this additional safety layer. Figure 11 shows what the RViz tools graphical user interface looks like when used with Franka Panda arm robot.

The autonomous driving of ARTI Chasi robot is achieved through Monte Carlo algorithm Simultaneous Localization and Mapping (SLAM), ROS navigation stack and Google Cartographer (<https://google-cartographer-ros.readthedocs.io/en/latest>, accessed on 8 May 2023). The robot uses light detection and ranging (LIDAR) and 3D cameras for indoor navigation. We applied existing outdoor navigation algorithms to work for indoor navigation in corridors and rooms.

Figure 12 shows a RViz visualisation of a sample robot navigation within the map of the Salzburg Research lab. On the left image one can see the robot in the lab trying to find his way down the narrow corridor to reach the room next to the lab (on the right side of the left image). On the right image we can see a clean bird perspective of the lab where the room on the right and the corridor have not yet been discovered by the cartographer. The food and drink pickup station is located in the lab (left room), the patient bed with Q.Bo social robot is situated in the neighboring right room.

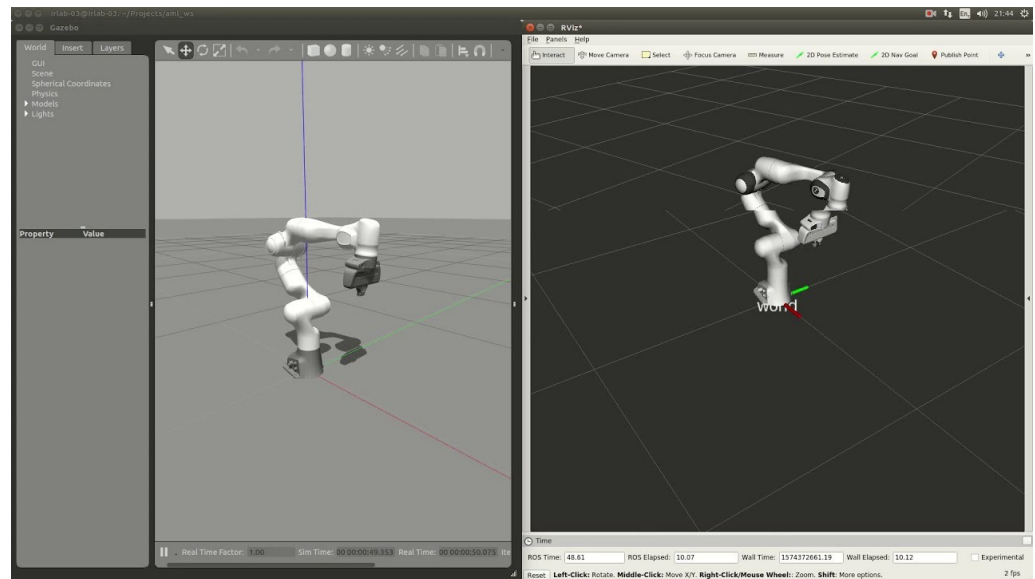


Figure 11. RViz tool layout-used for testing movement before execution.

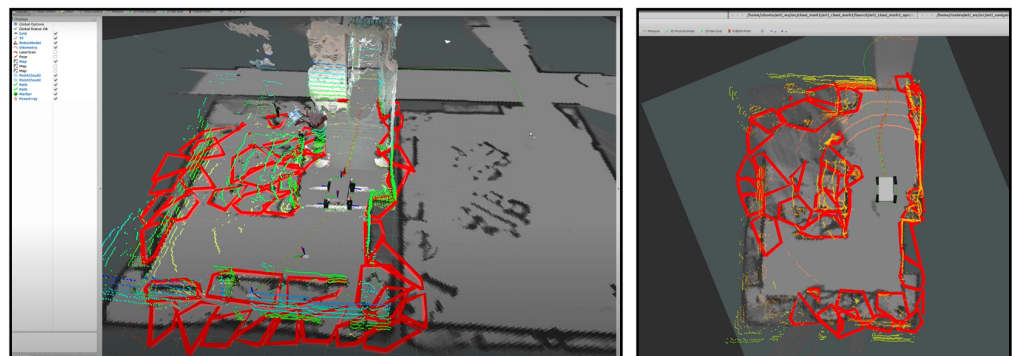


Figure 12. Arti Chasi driving-visualized with RViz tool.

However, this technical RTE toolchain is only of interest for robotic engineers. Task designers can simply use the final skills to implement appropriate tasks.

3.2. Use Case Industry (UCI)

The conceptualized industrial use case application targets the domain of manufacturing, specifically the pre-sorting, commissioning and delivery of finished parts or raw materials. Similarly, as for the healthcare domain, the primary focus is on applying the ROBxTASK programming approach to implement the required system behavior. Highly optimized primitives for, e.g., precise handling of parts, or visual perception are beyond the scope of this use case implementation. The following actors and sensors are integrated and coordinated within an OPC UA-based environment:

- UR10 CB3 (<https://www.universal-robots.com/products/ur10-robot>, accessed on 8 May 2023) (collaborative 6-DOF robot manipulator by Universal Robots). Implements the following skills: GrabObject (ObjectType, ObjectPosition), PutObject (TargetPosition), GetData (VariableName).
- MiR100 (<https://www.mobile-industrial-robots.com/solutions/robots/mir100>, accessed on 8 May 2023) (industrial mobile platform by Mobile Industrial Robots A/S). Implements the following skills: MoveToLocation (TargetLocation), GetData (VariableName).
- Peripherals including an electric gripper, a Sensopart VISOR (<https://www.universal-robots.com/plus/products/sensopart/visor-robotic>, accessed on 8 May 2023) camera system, and inductive/capacitive/light-barrier sensors.

Note that the mentioned peripherals (sensors and camera system) are physically linked and managed by the robot systems UR10 and MiR100. This is because of practical reasons, e.g., a robot controller usually comes with digital/analog I/O ports, and simplicity reason, i.e., to reduce system complexity.

The system layout considered for the industrial use case is depicted in Figure 13. The stationary UR10 robot system ('UCIUr10'), equipped with an electric gripper and a camera system, is configured to (a) grab objects of type 'Part_Cube' and 'Part_Cylinder' and (b) to put the same type of objects to predefined target locations. Those target locations include a 'Box', two slides (gravity chutes 'Slide.Part_Cube' and 'Slide.Part_Cylinder') are used as storage for sorted parts, and a tray to hold pre-commissioned parts of both types. The tray is mounted on an autonomously guided vehicle (AGV) of type MiR100 ('UCIMir'), which enables transport of parts to the required storage or production cell. Both slides include sensors at the entrance and bottom of the slides to enable interpretation of the storage level.

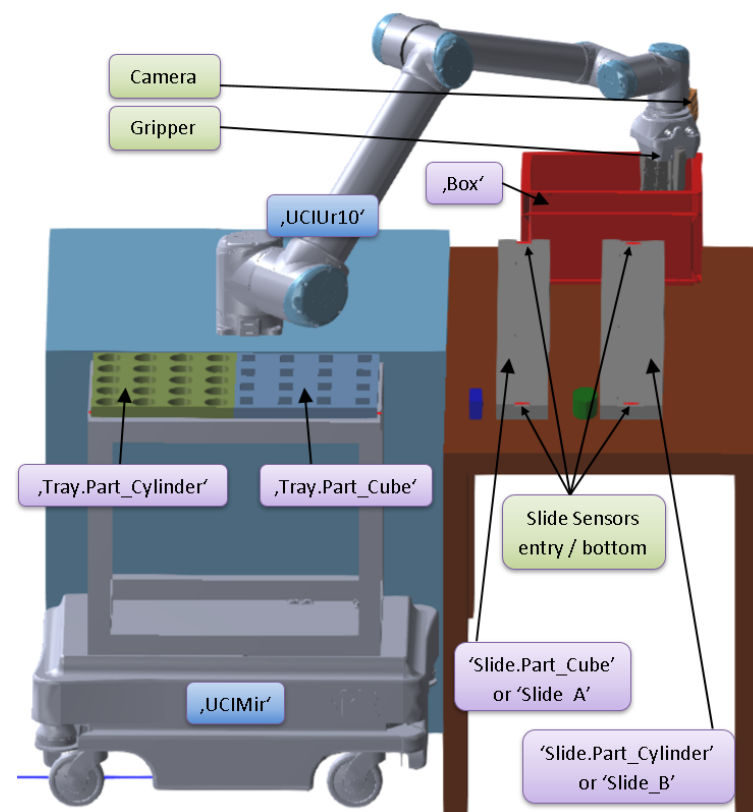


Figure 13. Schematic overview of the system layout of the industrial use case.

As an application use case, a three-step process shall be considered: (a) Bin-picking of parts (both types) out of the 'Box' and placing them onto the slide storage in a sorted fashion. (b) Picking sorted parts (both types) from the slide storage and commissioning them onto the tray at the designated tray-sections. (c) Transportation of the commissioned parts to the target destination.

The skills for UCI conform with the ROBxTASK skill definitions according to Table 1. To reduce implementation complexity for the task designer, the 'GrabObject' skill also includes the object localization process required to retrieve unordered parts from the 'Box'. Moreover, vision systems are tightly linked to the robot system as extrinsic calibration is often inevitable to achieve reasonable accuracy during part manipulation. The implementation of 'GrabObject' allows for an empty string passed to the 'ObjectType' slot if the type of object can be deduced from the specified 'ObjectPosition'. To give an example: 'Slide.Part_Cube' specified as position directly suggests that type 'Part_Cube' is of

interest. The slots 'ObjectPosition' and 'TargetPosition' also support alias-names for the slide storages. This need was identified during skill implementation, when initially the IDs 'Slide_A' and 'Slide_B' were introduced. However, while such identifiers can be intuitive enough for a robotic engineer, they are less self-explanatory for a task designer.

Using the setting described in Figure 13 we now wanted to research how to solve an industrial part delivery scenario. Figure 14 summarizes all available robots, skills, objects and locations. We recommend that such an overview graph also is provided to task designers so that they can understand all available robotic skills and their supporting arguments.

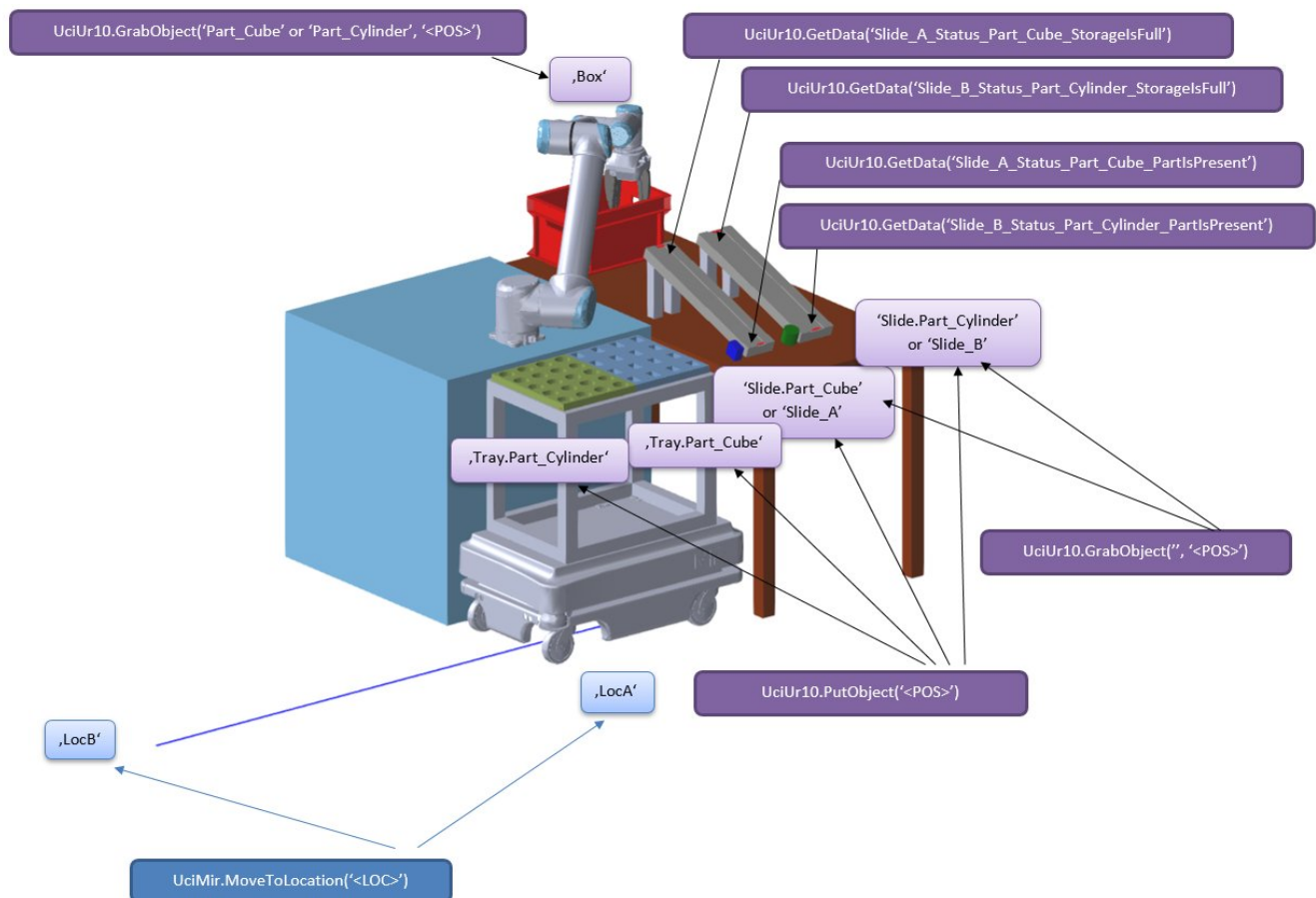


Figure 14. Robots, skills, objects and locations available in the industrial use case.

We divided the use case into two parts:

- **UCI1 'Fill Slides' (only UR10):** Task: Fill all slides available by picking parts (cubes and cylinders) from the box using UR10.
- **UCI2 'Deliver Parts' (UR10 + Mir100):** Task: MiR100 shall drive to LocA. Then 3 cubes and 5 cylinders shall be moved from the slides to the trays at the MiR100. Finally MiR100 shall drive to LocB, where these parts are needed. Coordination shall be done using messaging.

Figure 15 show a possible sample solution for UCI1 and Figure 16 a solution for UCI2.

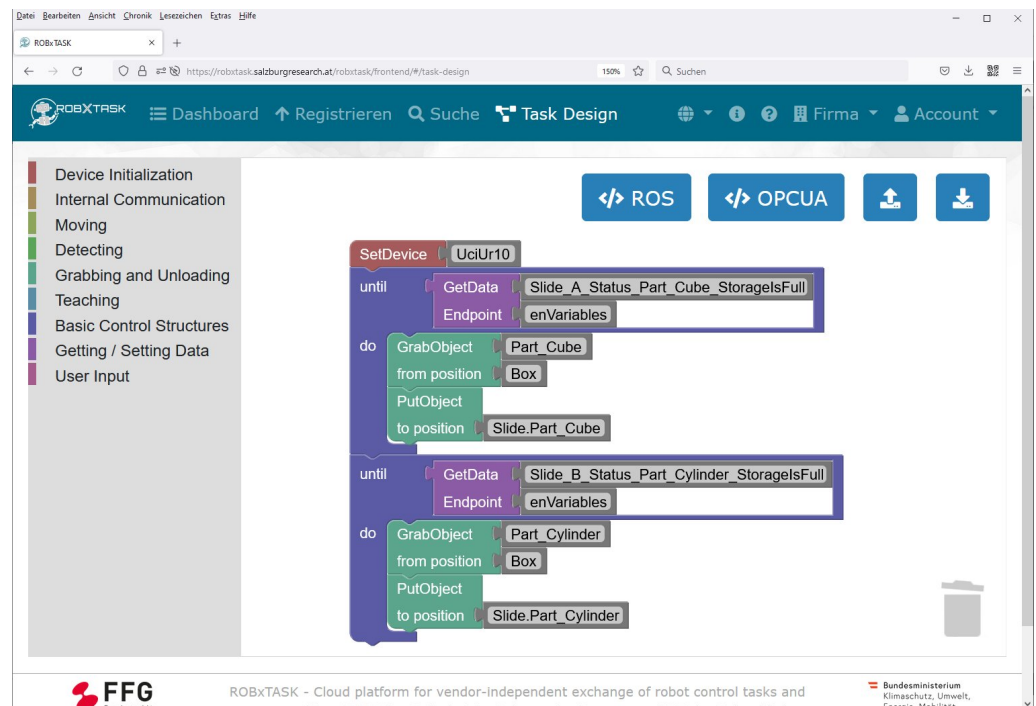


Figure 15. Sample Blockly code for UCI1 'Fill Slides'.

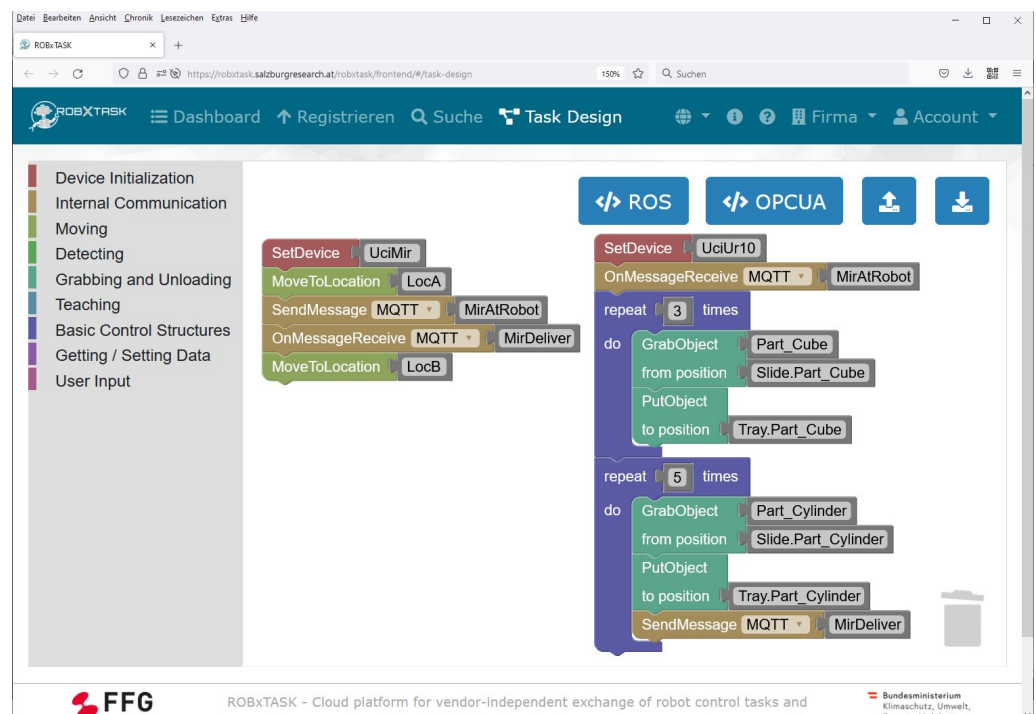


Figure 16. Sample Blockly code for UCI2 'Deliver Parts'.

3.2.1. Developing the Industrial Use Cases by Using OPC UA

For our UCI we first generated OPC UA servers that support the planned robotic skills by using XRob [59] and a SCXML based workflow designer tool.

Using some helper tools we then scanned the OPC UA servers to generate appropriate agent type stubs that can be used by agents to call the robotic skills using OPC UA, and the registration files for the ROBxTASK platform to register these skills. The code generator then was adapted to be able to call the appropriate software layer to support our RxRTE.

Now, when the Robot Co-Worker presses the button “</> OPCUA”, the flexible code generator generates Python code that supports RxRTE to download for all Blockly agents and these agents can then be started as individual processes.

In the project we can also generate mock objects [60], where skill calls are only simulated by pausing the program for 2 s. This feature is helpful for testing without robotic hardware and was used for our initial tests. Finally, a monitoring application was developed in the project, that enables users to follow the message exchange between devices and see the execution of the skills. A sample ROBxTASK monitor [19] webview of the running UCI2 process of Figure 16 is depicted in Figure 17.

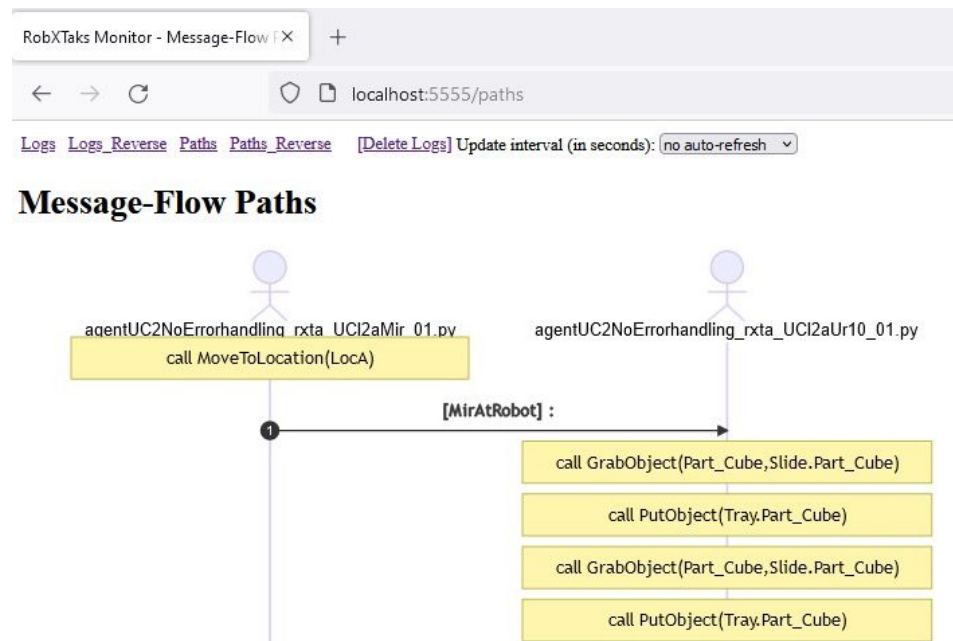


Figure 17. ROBxTASK monitor webview ‘Paths’ - displays parts of a sequence diagram for the use case industry scenario UCI2 of Figure 16.

Later both industrial use cases were also tested successfully by usage of the simulation system twin (<https://www.digifai.com/twin>, accessed on 8 May 2023) which was provided by our project partner Eberle Automatische Systeme GmbH & Co KG. twin is a physic based tool for the virtual commissioning of special purpose machines. In the project, e.g., a simulation model was built that works together with our robotic OPC UA servers for UCI (see Figure 18).

A video of the running simulation in twin that also shows the usage of the monitoring application is given in [61].

In the further progress of the project the scenario was then developed and tested using real hardware (see Figure 19). However, except for configuration changes, this did not mean any changes for the agents or the already developed code, since the OPC UA interface remained the same.

The simulation model developed in twin can also support task designers to test their robotic programs without a need for concrete hardware (see Section 4.3). Thus, within our project task designers can develop and test robotic workflows which are then transferable to run on real hardware.

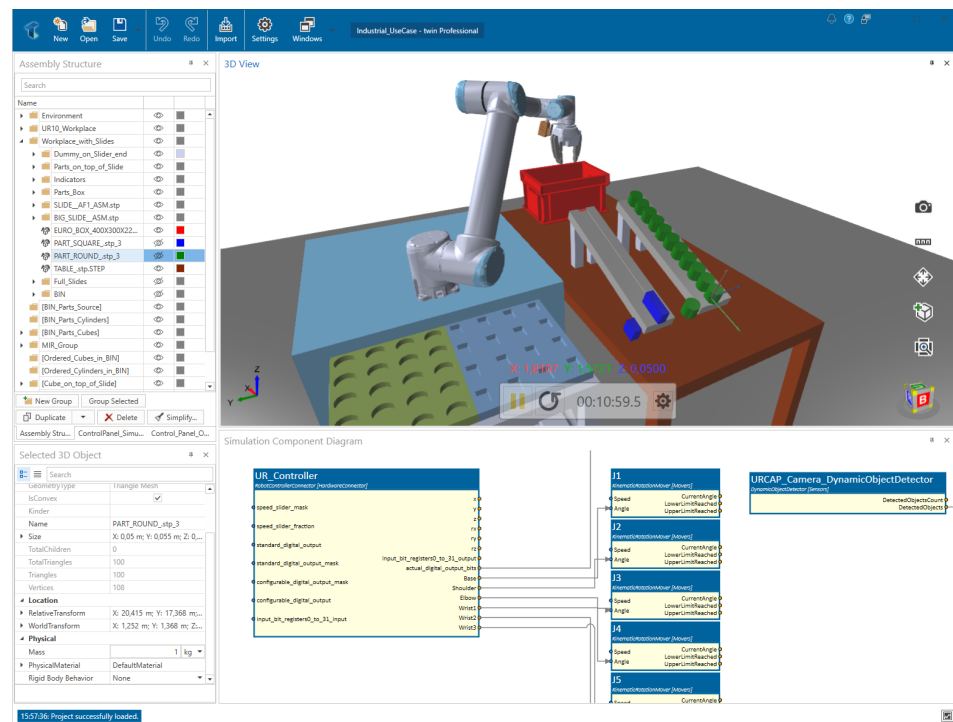


Figure 18. twin Setting for UCI.



Figure 19. Hardware Setting for UCI.

Discussions Concerning the UCI Scenarios

Although at first glance these scenarios seem simple. However, a closer look at the details raises some questions:

- UCI1: What if there are not enough parts in the box to fill the slides?

- UCI1: How must the program be changed if we want to work with 4 slides, for example?
- UCI2: What happens if there are not enough cubes or cylinders on the slides to pick them?
- UCI2: What if there is not enough space to put cubes or cylinders on the MiR100 trays?
- UCI2: What if MiR100 never sends a message?
- UCI2: What if the number of parts should be configurable?
- UCI2: Who starts the robotic workflow?
- UCI2: Shall the UR10 code end after servicing the Mir100, or wait for additional messages?
- UCI1, UCI2: What about error-handling?

Although error handling was not in the focus of this research project, we want to give some suggestions:

- Since we use code generators to generate the agent types, we can, for example, automatically add logging statements or exception handling if needed.
- Since each OPC UA skill is a state machine, the skill can be set to 'Halt' mode in case of errors and a robot co-worker can be informed to correct the problem. For any skill a 'StatusMessage' is available which holds the last problem text and can help the co-worker to solve the problem. For example, if skills can be "resumed correctly", the workflow can then continue. So, in this case, throwing an exception would not be a good idea, as this would then interrupt the entire workflow.
- We currently only use simple messages for coordination in the scenario UCI2. The RxRTE has some additional choreographic features which can be used if tasks shall run as autonomous services [19], but at the time of writing these features are not implemented as Blockly blocks for the platform.
- If configurability across multiple agents is required, some parameters can be sent between devices agents using message arguments. A second feature we evaluate at the RxRTE is to integrate the key-values data store solution REDIS [62] and then use scenario global variables instead. Setting global variables may be easier to understand for task engineers.

3.2.2. Environmental Skills-Virtual Storage Concept

UCI2 uses, e.g., the bottom slide-sensor of trays to find out if there is still space to put down parts. But what if we want to change the slides to an other kind of store concept later? Or what if the sensor-name differs on another slider scenario? An initial finding is that using real sensor-names in programs makes them less portable to other settings. Also if someone wants to enlarge the space for parts at the table, e.g., by adding additional slides 'Slide_C' and 'Slide_D' later, the programs then must be massively adapted by the task engineer. This might even be too complex.

Thus, we propose to hide these details for the task designer. Actually, the plan in the project was to include only the obvious skills of robotic devices via registration to the platform. But if we look deeper in this grab and put scenarios, we immediately see, that some kind of storage concept would be very useful for manipulators, boxes, slides and trays. If we grab objects we may have a camera at this position which can tell us if there's a part available to grab. Or some technical sensors could have this knowledge. Or some algorithm is internally counting parts on locations. One question in our use case was also whether to model the camera on the UR10 as an individual device. But we decided against this because the camera was mounted directly on the UR10. However, this now led to a sort of 'combined device' of a UR10 and a camera, which left room for discussion about what skills to provide then. For example, if someone implements only a skill *GrabObject* and internally hides the cameras skill *DetectObject*, then the error case when the camera did not detect a part must be included in the *GrabObject* skill.

However, if reuse and program simplicity are to be the primary concerns, we suggest that the task engineer not be burdened with these technical details. Instead we propose to add a "Virtual Storage" concept providing some appropriate skills:

At a minimum one storage skill shall give task designers the ability to count how many objects of a given objecttype are at a given location, e.g., *CountObjects(sObjType,*

sObjPosition). And a second skill, e.g., *CountObjectSpaces(sObjType, sObjPosition)* is needed to give the numbers of spaces available for a special object type at a given Position. Some additional comfort functions may, e.g., be *ExistsObject(sObjType, sObjPosition)* and *ExistsObjectSpace(sObjType, sObjPosition)*.

The usage of these skills now can increase the interoperability between different technical robot scenarios in the future by abstracting the concrete technical implementation. This also leaves enough room for the robotic engineer for later process improvements by, e.g., adding cameras, additional sensors or slides.

So in UCI1 instead of

```
1 while not UciUr10.GetData(Slide_A_Status_Part_Cube_StorageIsFull):
```

now

```
1 while UciUr10.CountObjectSpaces('Table', 'Part_Cube') > 0:
```

or even simpler

```
1 while UciUr10.ExistsObjectSpace('Table', 'Part_Cube'):
```

can be used to test if there's enough space left to put cubes on the (slide on the) table. And in both use cases *CountObjects* can be used before a grab operation to not grab parts if there are none available. Whereas *CountObjectSpaces* can prevent to try to put objects at positions where there is no space available, if this is needed in user programming.

Thus, in order to develop more robust, portable and adaptable tasks, we recommend determining the appropriate skills based on more than just the obvious robotic skills - we see a need for some kind of environmental skills too.

4. User Studies

In the context of developing interfaces for robot-programming by end-users, explicit emphasis of principles related to human centered design is called for [20]. Thus, the development of the ROBxTASK platform is embedded into an continuous human-centered design process involving various stakeholders from the industry and healthcare domains. In the following, we provide an overview of the methodology and results of user studies that have already been conducted in the context of the human-centered design process. Since the ROBxTASK platform is still under development, we also preview future studies aimed at evaluating refined versions of the platform.

4.1. Methodology

In the studies, we mainly focused on small scale usability tests as they are an efficient tool to identify usability issues at an early development stage. According to Faulkner [63], in a usability test involving only 5 participants 86% of usability problems can be identified on average. Each of our studies included task-based evaluations of the respective prototype employing the thinking aloud method [64] (i.e., participants were asked to solve various tasks with the platform while articulating their thoughts in the process) and additional interview questions.

So far, 3 user studies (RxT01, RxT02 and RxT03) have been carried out in the project. The studies were conducted in an iterative process that evaluated various prototypes of the platform and incorporated feedback from the studies into technical development. In each study, a refined version of the platform prototype was evaluated (compared to previous studies).

All studies involved stakeholders from industry and healthcare domains. Emphasis was put on early identification of obstacles related to the interaction with the platform and insights into possibilities for improvements. Due to restrictions related to the ongoing COVID-19 pandemic, studies were conducted in an online or hybrid setting. A video conferencing software was used for remote communication (speech and video) and recording of the study sessions (compliant with data protection regulations). Prior to the respective study, participants were informed about the goals and content of the study and gave their informed consent to participate.

While the first study did not include any training, training with the platform was applied for the second and third study. This means that before solving the study tasks, participants received a short briefing on how to use the platform. In addition, all persons who participated in the second study also took part in the third study to simulate a realistic environment in which persons become familiar with using the platform over time. Table 2 provides an overview of the sample, procedure and study tasks in the 3 different ROBxTASK studies conducted so far.

Table 2. ROBxTASK user studies overview.

	User Study RxT01	User Study RxT02	User Study RxT03
Sample	Females: 4, Males: 1 <i>Age</i> = 32.2	Females: 4, Males: 4 <i>Age</i> = 38.25	Females: 4, Males: 4 <i>Age</i> = 38.25
Procedure	Online	Online	Online or in Person
Training	No	Yes	Yes
Tasks	① Create an account. ② Add a new asset to a specific process. ③ Search for a new task module. ④ Edit and save a specific task module. ⑤ Recreate the described processes with the help of “skills” (blocks) provided (Industry domain). ⑥ Recreate the described processes with the help of “skills” (blocks) provided (Healthcare domain).	① Register a user account and a company. ② Define a process (using the task design editor) to exchange a message between two devices. ③ Define a process to grab and place an object. ④ Define a process in which a robot reacts to human speech input. ⑤ Define a process to move a robot to a different position. ⑥ Define a process for measuring dehydration in a patient. ⑦ Define a process to pick up and place multiple objects by a robot. ⑧ Freely define a process.	① Register an account or log in to the participant account. ② Search for devices. ③ Register a device. ④ Use the task design editor to define the process for the basic industrial use case. ⑤ Use the task design editor to define the process for the basic healthcare use case.

For analysis, recordings from the study sessions were transcribed. Based on this data, statements were grouped into overarching themes and implications for the further development of the platform were derived. The results presented in the following focus on positive and negative feedback from users and respective implications.

4.2. Results

4.2.1. User Study RxT01

In general, participants expressed positive feedback about the cloud platform mock-up in the first study. The design of the user interface was highly appreciated and all participants were able to complete the tasks using the VPL prototype, describing them as fairly simple. However, several participants suggested that initial training and additional documentation would be needed to make the platform even more accessible. All participants indicated that they could imagine using such a platform. More detailed results from the first (positive and negative feedback from participants) study are presented in Table 3.

Table 3. Feedback from participants of the user study RxT01.

Positive Feedback	Negative Feedback
<ul style="list-style-type: none"> Interaction with the user interface is accessible and rather easy to understand. Using the VPL prototype is fun. Design of the VPL prototype is pleasant. Blocks hovering help in the VPL prototype is appreciated. Blocks predefined options in the VPL prototype is useful. 	<ul style="list-style-type: none"> The registration page would need to be made more user friendly (e.g., option to see password, password guideline, link to End User License Agreement). Dashboard screen terminology is unclear at first. <i>Account</i> and <i>Company</i> tabs can be combined. <i>Configure</i> functionality is unclear. Some icons could be improved (e.g., <i>Download</i>, <i>Search</i>). For some functionalities (e.g., <i>Configure</i>, <i>Download</i>) the participants don't understand whether users items or the online version are meant. The meaning of the <i>Skills</i> and <i>SupportedModules</i> assets is unclear. On the <i>TaskDesign</i> screen the difference between <i>Blocks</i>, <i>TaskModules</i> and <i>Processes</i> is unclear. Interaction with the <i>TaskDesign</i> screen is a little bit confusing. For experienced programmers the VPL prototype is too simple and does not offer enough flexibility. Initial training and documentation is required.

A list of implications was derived from the analysis of the participants comments and suggestions. Each implication was assigned a priority “high”, “middle” or “low”. These implications will be considered as the platform continues to evolve. For example, implications rated as “high” focused on maintaining a consistent tab layout and terminology, defining some functionalities more clearly, displaying more information when hovering, necessity for documentation, grouping blocks into categories. The implications also contributed to the improvement of the studies, e.g., the necessity for documentation and training was rated as “high” and therefore a training concept was developed for the following studies.

4.2.2. User Study RxT02

Generally, the second user study showed positive feedback from the participants from both the healthcare and the industry domains. The participants appreciated the design of the platform prototype and the simplification of programming. Participants highlighted the benefit of initial training to get an introduction to the platform functionalities and the programming. They also showed interest in using the platform in the future. In addition, participants pointed out some negative aspects of the platform and future improvements they would like to see. Participants from both areas commented as follows on the tasks they had to complete during the assessment:

- Interaction with user registration on the platform was considered very easy.
- The tasks that focus on an action that requires only a few blocks, such as exchanging messages, grabbing an object, or moving to a position, were considered easy to perform and required minimum assistance from the study facilitator.
- Tasks requiring the design of a complete process were considered more challenging and required more support in interpreting the task and translating the task into blocks.

The feedback is presented in more detail in Table 4.

Table 4. Feedback from participants of the user study RxT02.

	Positive Feedback	Negative Feedback
Healthcare Domain	<ul style="list-style-type: none"> Platform is easy to use and to understand. Programming is simple and fun with the task design. Thanks to the blocks and the drag and drop feature. Users without technical knowledge can imagine using the platform in the future. Platform could be interesting for other sectors such as schools or training. 	<ul style="list-style-type: none"> For non-programmers some information or explanations are missing (e.g., what is a string object). Better information or explanation for blocks is missing (e.g., some blocks seem to have the same function and for some blocks participants don't understand how to use them). To make the platform more practical, some functions should be improved (select multiple blocks together, scroll function in the <i>TaskDesign</i> editor). More information for blocks, command and devices would be useful. Lack of feedback from the platform on whether the user is using the blocks as intended.
Industry Domain	<ul style="list-style-type: none"> Platform design is very pleasing. Programming is simplified and easy to handle for a first time user thanks to the existing blocks and the hovering help. Platform could be used for training purposes and show many robotics applications. 	<ul style="list-style-type: none"> Some blocks need more information to use them (e.g., data unit and format). Blocks need some optimizations to make the programming more natural (e.g., use blocks inside blocks, blocks prefilled with example). Saving and loading of a program is not user-friendly. Make the saving/loading of a program more standard. Make a short tutorial included in the platform for beginners. More information about blocks.

In the second study, the implications focused on specific aspects of certain functionalities. For example, the “Save” button should directly save the user’s program as an XML file and not open the XML code in another browser tab, the “Load” button should open the Windows Explorer instead of providing an interface for copying and pasting the XML file, the type of the blocks’ parameters should be clearer, access to the free text field should be easier, or more information should be added when hovering.

4.2.3. User Study RxT03

In the third study, the less complex tasks (e.g., logging in) were considered easy-except for the registration of a device. In contrast, the more complex tasks involving the task design editor (i.e., designing complete use cases for health and industry domains) required the assistance of the study facilitator and were more challenging for the participants. Detailed feedback from participants from the third study is presented in Table 5.

Table 5. Feedback from participants of the user study RxT03.

	Positive Feedback	Negative Feedback
Healthcare Domain	<ul style="list-style-type: none"> Task design presentation is easy to understand. Task design requires some time to get used to, but can be operated independently after training. 	<ul style="list-style-type: none"> Password restoration does not always work. Search function needs the device full name. Usage of a JSON file for the device registration cannot be used by non-programmers. Free text field should be accessed in a faster more convenient way. Drag and drop feature could be optimized to insert multiple blocks at once. Common keyboard shortcuts do not work (e.g., CTRL + A) in the <i>TaskDesign</i>.
Industry Domain	<ul style="list-style-type: none"> Password registration works well. Search function filters the devices. User's devices are easy to find (only devices users can delete or edit in the list). 	<ul style="list-style-type: none"> Users do not understand what they can do with the device found via the search function. There is then only information to read. Registration of a device is too complicated with the JSON file. There is no indication if the user made an error in the JSON file. In the "if loop" there is no "else" possibility available. Some parameters of the blocks are unclear. Switching from the <i>TaskDesign</i> tab to another tab erases the user's program. Differences between some blocks are not always clear.

Since the participants in the third study already had more experience with the platform (see Section 4.1), the implications derived from the third study focused primarily on the newly introduced functionalities. For example, device skills could be more accurately described in search results, device registration could be made simpler (no use of a JSON file), runtime errors should be fixed, and the user account password restoration could be improved. Finally, the fact that most participants required help from the study facilitators when designing more complex use cases again shows that while the platform is considered easy to use by most users—approaches for end-user training are still important to consider.

As part of the third user study (in addition to the evaluation with potential end-users), a checklist of 27 functional criteria (i.e., functionalities that should eventually be implemented in the final version of the platform) was created and evaluated by two study facilitators. Based on the checklist, the current prototype of the platform was explored and the individual criteria were reviewed. The results are depicted in Table 6 and provide an comprehensive overview of the functionalities implemented in the current ROBxTASK prototype.

Table 6. ROBxTASK platform functional criteria (✓ fulfilled, □ partly fulfilled, ✗ not fulfilled, ? not yet tested).

Group	Criteria
Platform	<ul style="list-style-type: none"> ✓ Users can log in to the platform using their name and password. ✓ A new user account can be created on the cloud platform. □ (New) devices can be added to a process in the cloud platform. Actual status: JSON Files only. □ Processes defined using blocks or task modules can be saved in the cloud platform. Actual status: JSON Files only, download button missing. ✓ Operations in the task design editor can be undone (optional). ✗ Operations in the task design editor can be repeated (optional). □ Task modules, processes and devices (assets) can be searched for in the cloud platform. Actual status: only available for some devices. ✗ A flag for public access (i.e., public or private) can be set when creating/saving task modules or processes to allow access only for certain user roles on the platform ✗ If the access to a task module or process is set to private (i.e., non-public) only certain user roles can view and access the respective item.
Programming	<ul style="list-style-type: none"> ✓ All blocks or task modules required for implementing the basic use cases are available in the cloud platform. ✓ All devices required to implement the basic use cases are available in the cloud platform. ✓ Different blocks or task modules can be combined with each other in the cloud platform (“workspace” of the task design editor) using drag and drop. ✓ Blocks or task modules are organized into categories in the task design editor. ✓ Blocks or task modules can be removed from the task design editor workspace. ✓ Blocks, task modules and combinations of them can be copied from the task design editor workspace. Usability improvements required. ✓ Blocks, task modules and combinations of them can be cut from the task design editor workspace. Usability improvements required. ✓ Blocks, task modules and combinations of them can be pasted from the clipboard into the task design editor workspace. ✗ Blocks or task modules can only be assigned to a device if that device can execute them. ✓ Arguments can be passed to blocks and task modules in the cloud platform. □ Predefined variables required for the implementation of the use cases (e.g., position specifications) are defined in the cloud platform. ✗ Communication between different devices is visualized in the task design editor. □ Additional information on the functionality of a block or task module is displayed when hovering over the item in the task design editor.
Runtime	<ul style="list-style-type: none"> □ Program code is generated by combining blocks or task modules in the task design editor. ✓ Program code generated by combining blocks or task modules can be downloaded or exported. ? Downloaded or exported program code can be transferred to the simulation environment. ? Code transferred to the simulation environment correctly represents the use case for the industry domain and can be executed without errors. ? Code transferred to the simulation environment correctly represents the use case for the healthcare domain and can be executed without errors.

4.3. Outlook

An important aspect raised by the test users was that they also wanted to see the robotic actions when executing their programs. Of course, this is not easy to solve with real hardware. But in a later stage of the project we developed a docker based testing environment where our usecases UCH and UCI can be tested virtual by help of the twin models that were developed in earlier stages of the project. Users can (1) download their robotic programs from the ROBxTASK platform to some shared folder of a docker-container,

(2) start the program within a docker-container, and (3) see the simulation in the 3D-view of twin while (4) observing the monitoring output of the distributed workflow (see Figure 20).

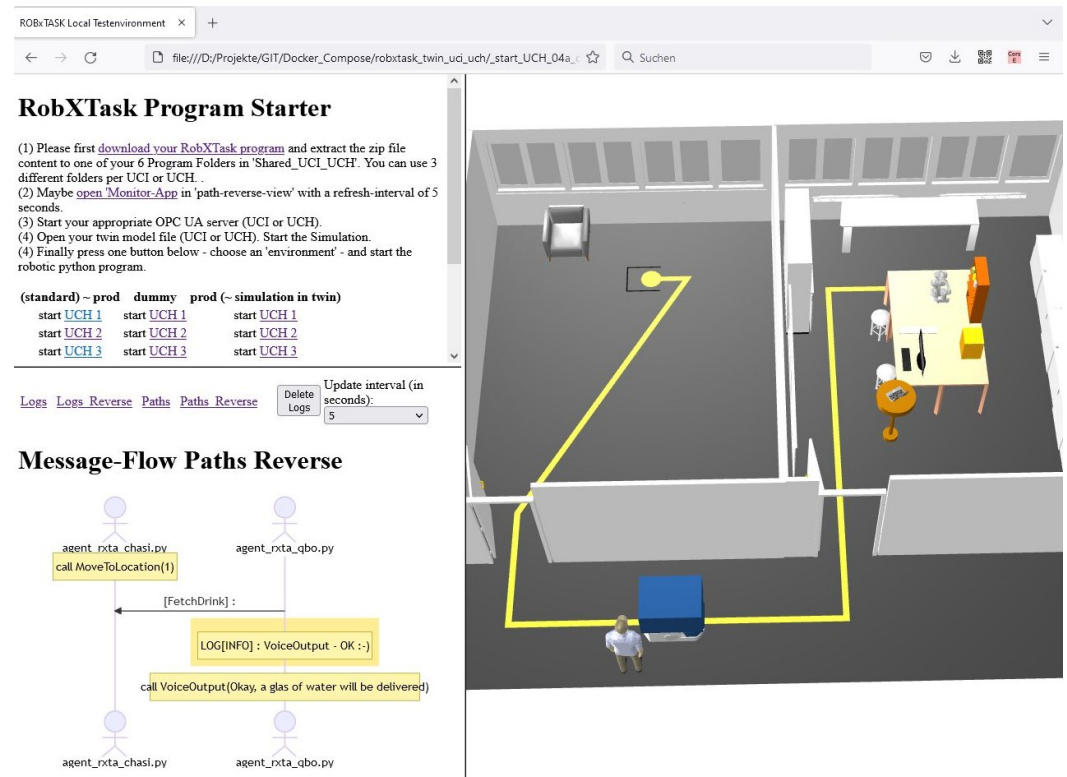


Figure 20. Screenshot of ROBxTASK End-User Testing Environment for the Healthcare Use Case.

An example video that shows how this testing environment can be used by end-users is given in [65].

The upcoming user studies will use the new testing environment, and focus on evaluating revised and improved, as well as newly implemented, functions of the platform and the VPL. The studies will be based on the healthcare and industrial use cases defined in the ROBxTASK project and will collect objective metrics (e.g., task time) and subjective data (e.g., ratings on user experience, qualitative feedback) to continuously monitor and improve the quality of the application. Furthermore, future studies will evaluate the platform in real-world settings and explore factors that may facilitate or hinder the practical application of the task design process from an end-user perspective.

5. Discussions

By leveraging NIMBLE's [51] existing technology stack, the ROBxTASK platform could be faster developed than starting from scratch. With the help of the flexible architecture, further improvements can be added more easily in the future (e.g., self-registering of devices).

In the project a flexible code generator was developed that allows to generate, e.g., Python code for different RTEs. For our healthcare use case the code generator, e.g., generates code to call ROS actions. Whereas for the industrial use case the code generator produces source code to be run by using a RxRTE that was developed to be able to also support exotic runtime environments and robotic devices. The ROBxTASK platform can therefore be used with various robotic RTEs and thus, the concrete RTE used can be hidden for the task designers.

In our industrial use case we contact the robotic devices via OPC UA. For the OPC UA environment, we also developed a parser and code generator so that as new skills become available on a suitable robotic OPC UA server, we can generate agent type stubs and JSON

files to register these skills on the platform and then let task designers use them, speeding up development.

The logging and monitoring capabilities of the RxRTE environment proved to be helpful in observing distributed tasks. Displaying message flow paths enables robotic co-workers to comfortably follow the current state of skill execution. Even task designers can pre-test robotic workflows to find problems in a workflow, when using a mocking environment and RxRTE. The platform currently supports only basic interoperability (messaging) to be used between agents for the ROBxTASK platform. Further improvements that support service discovery and mediation have been implemented in the RxRTE [19] but are currently not available using Google Blockly blocks to be used in the platform. In general additional RxRTE features added later, can also be registered on the platform to provide additional blocks for task programming. Since AI capabilities, database access and other additional functions can be provided in this way, the development of more intelligent and autonomous software agents shall be supported in the future.

By integration twin simulation, problems in the environment can now be more easily detected by robot co-workers before they work on the real hardware. Using twin for testing is also helpful for task designers to test the robotic programs and improve their understanding on how to properly use the robotic skills. But of course, this requires manual development of a suitable simulation environment.

If advanced error handling is to be integrated in the future, more research is needed. In this context, we see maintaining usability for task designers while developing robust robot workflows as a challenge yet to be solved.

6. Conclusions

In the research project ROBxTASK, the partners developed a platform to support end-users in designing and executing robotic processes. Using the platform, multiple user groups work together. Robotic engineers can register (robotic) devices including their skills by using appropriate JSON files. Task designers can use these skills in the “Google Blockly”-based, graphical task design editor to create workflows across multiple robotic agents. And robot co-workers can download and execute the workflow code in a local environment with real robots or digital twins. The platform’s operational capability was evaluated by end users based on healthcare and industrial use cases.

In workshops MIRO boards proved to be useful to identify skills at the right level of abstraction to be used by task designers. These skills were then applied to use cases in the industrial and healthcare domain. The project used the same skills in different domains and implemented them through different robotic devices to demonstrate the usability of the skills across domains and devices.

In ROBxTASK we have created the three roles described above that collaborate to enable end users without robotic expertise to focus on their tasks, while being able to get support from robotic engineers. Application specific workflows are used to integrate heterogeneous systems (including robots). Individual tasks in these workflows are realized by skills, implemented in the technical systems. In order to develop more robust, portable and adaptable tasks, we recommend determining the appropriate skills based on more than just the obvious robotic or device skills. For example, using “Environmental skills” according to our “Virtual Storage” concept proposed in Section 3.2.2, can help to develop more stable grasping/lying programs.

In the project three user studies were carried out. The user studies showed that although the platform is relatively easy to use, a training concept for end users is necessary. Tasks that focus on an action that requires only a few blocks, such as exchanging messages, grabbing an object, or moving to a position, were considered easy to perform but design of a full process is considered more challenging.

However, participants generally indicated that they could imagine using such a platform, so the platform was considered interesting.

Overall, the ROBxTASK platform is able to help organisations to integrate robotic systems along a workflow. Through features like sharing skills and workflows the platform enables knowledge exchange even across multiple organizations. The three user roles that have been created for ROBxTASK allow that individuals focus with their expertise on certain tasks, where, most importantly, end users can work in their domain and make use of robots without much training. Also the robotic programmer can focus on implementing skills, without the need to understand in detail the application domain(s). Only the role of the process designer needs knowledge of both worlds. However, that role receives the most support from the implemented tools. The graphical process editor allows to stay on a higher level of abstraction with respect to robotic know-how.

The ability to provide simulations for end-user testing proved to be a great added value for end users. However, developing individual simulation environments for each robot environment is time consuming. To this end, some ideas have already been discussed by the project participants to possibly create suitable environments more automatically in a future follow-up project. Further research has to be taken especially when e.g., advanced error handling shall be integrated. Here, ease of use for task designers while building robust robotic workflows can be a challenge.

In the next steps of the project, further studies will be conducted to evaluate the platform with end users in real-world environments.

Author Contributions: Conceptualization, H.Z., M.S.T., T.B. and G.W.; methodology, H.Z., M.S.T., G.W., T.B., N.S., M.P., G.S., F.S., A.S. and C.N.; software, M.S.T., M.P., C.N., D.S. and H.Z.; validation, G.W. and H.Z.; formal analysis, N.S., T.B., G.S. and C.N.; investigation, N.S., T.B., G.S. and C.N.; resources, M.S.T., F.S., C.N., T.B. and M.P.; writing—original draft preparation, H.Z., M.S.T., G.W., T.B., N.S., M.P., G.S., F.S., A.S. and C.N.; writing—review and editing, G.W., A.S., H.Z. and A.P.; project administration, M.S.T. and F.S.; funding acquisition, F.S., G.W., M.S.T., A.S., N.S., M.E., C.N. and A.P.; principle investigators F.S. and G.W. All authors have read and agreed to the published version of the manuscript.

Funding: The research was partially funded by the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) and the Federal Ministry for Digital and Economic Affairs (BMDW) through the Austrian Research Promotion Agency (FFG) in project ROBxTASK (Prj. Nr.: 880866).

Institutional Review Board Statement: Ethical review and approval were waived for this study due to experience from a similar projects (e.g., RoboGen, PNr. 866694, AT programme FEMtech), for which ethical commissions clarified non-necessity of approval at this development stages. In the project ROBxTASK ethical monitoring of the requirements regarding ethical, privacy and data protection issues throughout the project lifetime will be achieved by means of designated audit forms and questionnaires (e.g., MEESTAR analysis). This task will also examine the extent to which the project solutions affect ethical, privacy and data protection issues.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AGV	Autonomously Guided Vehicle
AMR	Autonomous Mobile Robot
B2B	Business to Business
COVID-19	Coronavirus SARS-CoV-2
DOF	Degree Of Freedom
IFR	International Federation of Robotics
IFTTT	If This Then That
LIDAR	Light Detection and Ranging

MEESTAR	Model for the ethical evaluation of socio-technological arrangements
MDPI	Multidisciplinary Digital Publishing Institute
M2M	Machine to Machine
NIMBLE	collaboration Network for Industry, Manufacturing, Business and Logistics in Europe
OPC UA	OPC Unified Architecture
RTE	Runtime Environment
ROS	Robot Operating System
RViz	ROS visualization
RxRTE	ROBxTASK RTE
SCXML	State Chart XML
SLAM	Simultaneous Localization and Mapping
SOP	Standard Operation Procedures
SRFG	Salzburg Research Forschungsgesellschaft
UCI	Use Case Industry
UCH	Use Case Healthcare
VPL	Visual Programming Language
XML	Extensible Markup Language

Appendix A. Sample Dialogs for the ROBxTASK Platform

The ROBxTASK platform is based on the existing technology stack of (NIMBLE) [51]. Therefore, a lot of documentation can already be found online on the NIMBLE documentation website (<https://www.nimble-project.org/docs/>, accessed on 8 May 2023).

In addition, only some of the most important ROBxTASK specific dialogs that task designers come into contact with in connection with the development of robotic programs are shown below.

After successfully login to the ROBxTASK platform, the initial dashboard shown in Figure A1 is presented. Users then can register devices or start writing programs using the *TaskDesign* editor.

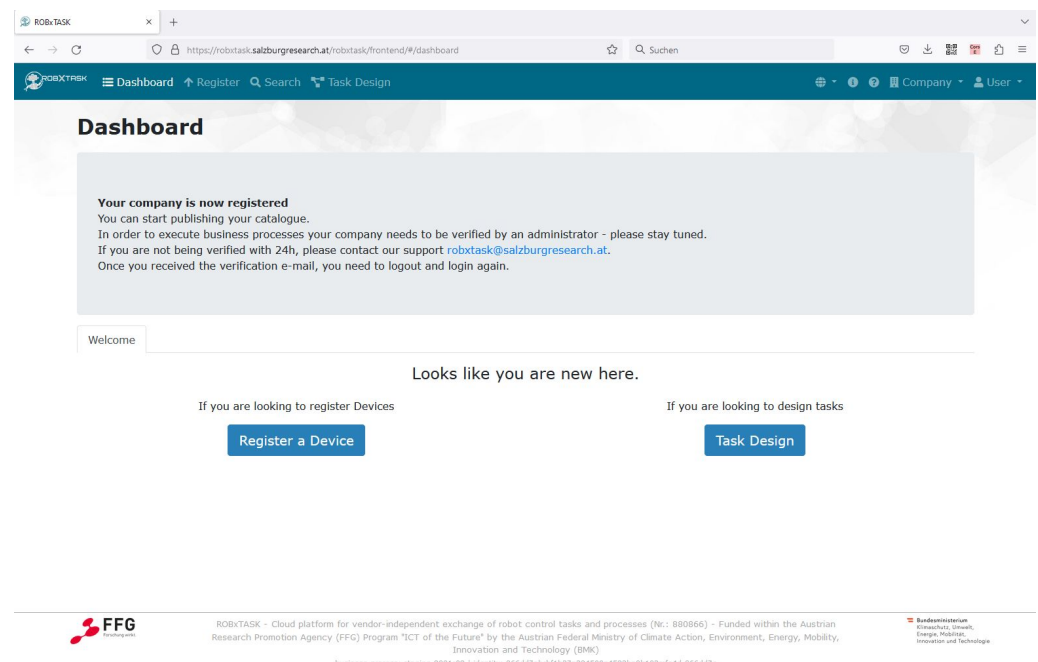
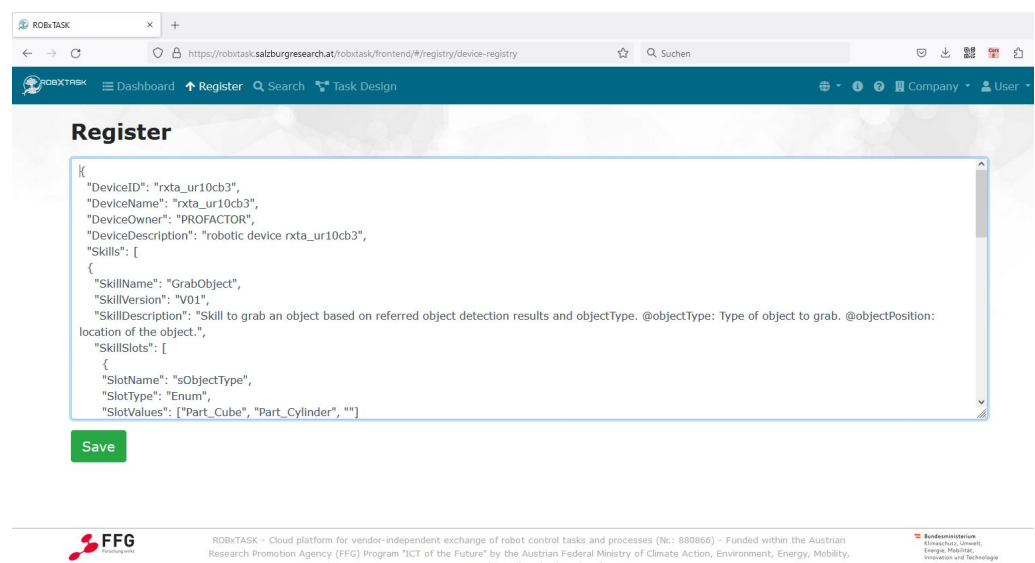


Figure A1. ROBxTASK platform initial dashboard after successfully login.

Robotic Engineers can register new devices using JSON based registration file containing all available skills of the target robot by using the form depicted in Figure A2. This architecture also enables the registration of devices that are currently not yet available in this form, which in turn enables initial usability tests before the corresponding skills have to be developed in simulation or reality.



```

{
  "DeviceID": "rxta_ur10cb3",
  "DeviceName": "rxta_ur10cb3",
  "DeviceOwner": "PROFACTOR",
  "DeviceDescription": "robotic device rxta_ur10cb3",
  "Skills": [
    {
      "SkillName": "GrabObject",
      "SkillVersion": "V01",
      "SkillDescription": "Skill to grab an object based on referred object detection results and objectType. @objectType: Type of object to grab. @objectPosition: location of the object.",
      "SkillSlots": [
        {
          "SlotName": "sObjectType",
          "SlotType": "Enum",
          "SlotValues": ["Part_Cube", "Part_Cylinder", ""]
        }
      ]
    }
  ]
}

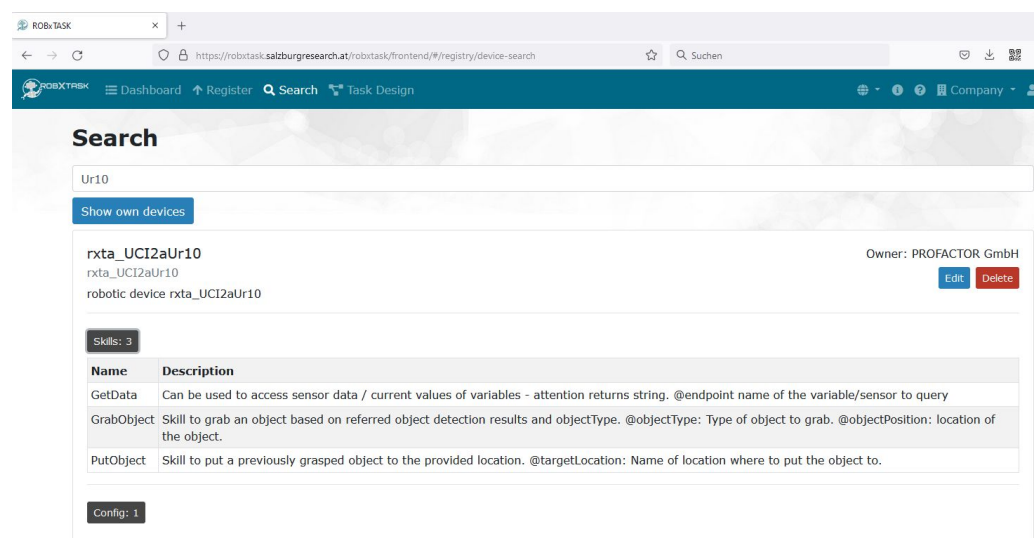
```

Save

FFG
Research Promotion Agency (FFG) Program "ICT of the Future" by the Austrian Federal Ministry of Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK)

Figure A2. Form to register a device in the ROBxTASK platform.

All registered devices can be searched by task designers, to use them for their robotic workflows (see Figure A3). The result of the query explains the available devices, their skills and parameters.



Ur10

Show own devices

rxta_UCI2aUr10
rxta_UCI2aUr10
robotic device rxta_UCI2aUr10

Owner: PROFACTOR GmbH

Skills: 3

Name	Description
GetData	Can be used to access sensor data / current values of variables - attention returns string. @endpoint name of the variable/sensor to query
GrabObject	Skill to grab an object based on referred object detection results and objectType. @objectType: Type of object to grab. @objectPosition: location of the object.
PutObject	Skill to put a previously grasped object to the provided location. @targetLocation: Name of location where to put the object to.

Config: 1

Figure A3. Dialog to search for registered devices in the ROBxTASK platform.

References

1. IFR. International Federation of Robotics-World Robotics 2021 Industrial Robots Report. Available online: <https://ifr.org/ifr-press-releases/news/robot-sales-rise-again> (accessed on 19 July 2022).
2. Csefalvay, Z.; Gkotsis, P. *Global Race for Robotisation-Looking at the Entire Robotisation Chain*; EUR 30311 EN; Publications Office of the European Union: Luxembourg, 2020; ISBN 978-92-76-20875-4. [CrossRef]
3. Raj, R.; Kos, A. A Comprehensive Study of Mobile Robot: History, Developments, Applications, and Future Research Perspectives. *Appl. Sci.* **2022**, *12*, 6951. [CrossRef]
4. Wood, L. Global AGV (Automated Guided Vehicles) and AMR (Autonomous Mobile Robots) Market Forecast to 2026: Contains Analysis of More Than 500 Players. 2021. Available online: <https://www.globenewswire.com/news-release/2021/01/13/2157658/0/en/Global-AGV-Automated-Guided-Vehicles-and-AMR-Autonomous-Mobile-Robots-Market-Forecast-to-2026-Contains-Analysis-of-More-Than-500-Players.html> (accessed on 29 July 2022).
5. Holland, J.; Kingston, L.; McCarthy, C.; Armstrong, E.; O'Dwyer, P.; Merz, F.; McConnell, M. Service Robots in the Healthcare Sector. *Robotics* **2021**, *10*, 47. [CrossRef]
6. Jahn, U.; Heß, D.; Stampa, M.; Sutorma, A.; Röhrig, C.; Schulz, P.; Wolff, C. A Taxonomy for Mobile Robots: Types, Applications, Capabilities, Implementations, Requirements, and Challenges. *Robotics* **2020**, *9*, 109. [CrossRef]

7. Duffy, B.R.; Rooney, C.; O'Hare, G.M.; O'Donoghue, R. What is a social robot? In Proceedings of the 10th Irish Conference on Artificial Intelligence & Cognitive Science, University College Cork, Ireland, 1–3 September 1999.
8. Youssef, K.; Said, S.; Alkork, S.; Beyrouthy, T. A Survey on Recent Advances in Social Robotics. *Robotics* **2022**, *11*, 75. [\[CrossRef\]](#)
9. Weichhart, G.; Panetto, H.; Gutiérrez, A.M. Interoperability in the Cyber-Physical Manufacturing Enterprise. *Annu. Rev. Control* **2021**, *51*, 346–356. [\[CrossRef\]](#)
10. Fülöp, M.T.; Gubán, M.; Gubán, A.; Avornicului, M. Application Research of Soft Computing Based on Machine Learning Production Scheduling. *Processes* **2022**, *10*, 520. [\[CrossRef\]](#)
11. Zdravković, M.; Panetto, H.; Weichhart, G. AI-enabled Enterprise Information Systems for Manufacturing. *Enterp. Inf. Syst.* **2021**, *16*, 668–720. [\[CrossRef\]](#)
12. Thalmann, S.; Mangler, J.; Schreck, T.; Huemer, C.; Streit, M.; Pauker, F.; Weichhart, G.; Schulte, S.; Kittl, C.; Pollak, C.; et al. Data Analytics for Industrial Process Improvement A Vision Paper. In Proceedings of the 2018 IEEE 20th Conference on Business Informatics (CBI), Los Alamitos, CA, USA, 11–14 July 2018; Volume 1, pp. 92–96. [\[CrossRef\]](#)
13. Weichhart, G.; Mangler, J.; Mayr-Dorn, C.; Egyed, A.; Hämmerle, A. Production Process Interoperability for Cyber-Physical Production Systems. *IFAC-PapersOnLine* **2021**, *54*, 906–911. [\[CrossRef\]](#)
14. Weichhart, G.; Mangler, J.; Raschendorfer, A.; Mayr-Dorn, C.; Huemer, C.; Hämmerle, A.; Pichler, A. An Adaptive System-of-Systems Approach for Resilient Manufacturing. *e & i Elektrotechnik Informationstechnik* **2021**, *138*, 341–348. [\[CrossRef\]](#)
15. Fülöp, M.T.; Udvaros, J.; Gubán, A.; Sándor, A. Development of Computational Thinking Using Microcontrollers Integrated into OOP (Object-Oriented Programming). *Sustainability* **2022**, *14*, 7218. [\[CrossRef\]](#)
16. Weichhart, G.; Pichler, A.; Wögerer, C. Workflow Representations for Human and Artificial Agent Collaborations. In Proceedings of the ACHI 2018, the Eleventh International Conference on Advances in Computer-Human Interactions, Rome, Italy, 25–29 March 2018; pp. 132–135.
17. Weichhart, G.; Pichler, A.; Strohmeier, F.; Schmoigl, M.; Zörrer, H. The ROBxTASK architecture for interoperability of robotic systems. In Proceedings of the 2021 IEEE International Workshop on Metrology for Industry 4.0 and IoT, Rome, Italy, 7–9 June 2021; pp. 449–453. [\[CrossRef\]](#)
18. Bieg, T.; Schmoigl-Tonis, M.; Sturm, N.; Nativel, C.; Sackl, A. Enabling Cross-Domain Robot Programming by End-Users: The ROBxTASK Platform. In Proceedings of the 2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Roma, Italy, 12–16 September 2022; pp. 1–3.
19. Zörrer, H.; Propst, M.; Weichhart, G.; Pichler, A.; Strohmeier, F.; Schmoigl-Tonis, M. ROBxTASK RTE—A lightweight runtime environment to implement collaborative processes across different robotic systems. *IFAC-PapersOnLine* **2022**, *55*, 2647–2652. [\[CrossRef\]](#)
20. Kuhail, M.A.; Farooq, S.; Hammad, R.; Bahja, M. Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. *IEEE Access* **2021**, *9*, 14181–14202. [\[CrossRef\]](#)
21. Leonardi, N.; Manca, M.; Paternò, F.; Santoro, C. Trigger-Action Programming for Personalising Humanoid Robot Behaviour. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, New York, NY, USA, 4–9 May 2019; CHI '19, pp. 1–13. [\[CrossRef\]](#)
22. Buchina, N.; Kamel, S.; Barakova, E. Design and evaluation of an end-user friendly tool for robot programming. In Proceedings of the 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), New York, NY, USA, 26–31 August 2016; pp. 185–191. [\[CrossRef\]](#)
23. Zubrycki, I.; Granosik, G. Designing an interactive device for sensory therapy. In Proceedings of the 2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Christchurch, New Zealand, 7–10 March 2016; pp. 545–546. [\[CrossRef\]](#)
24. Barakova, E.I.; Gillesen, J.C.; Huskens, B.E.; Lourens, T. End-user programming architecture facilitates the uptake of robots in social therapies. *Robot. Auton. Syst.* **2013**, *61*, 704–713. [\[CrossRef\]](#)
25. Jost, B.; Ketterl, M.; Budde, R.; Leimbach, T. Graphical programming environments for educational robots: Open Roberta—yet another one? In Proceedings of the 2014 IEEE International Symposium on Multimedia, Taichung, Taiwan, 10–12 December 2014; pp. 381–386. [\[CrossRef\]](#)
26. Coronado, E.; Mastrogiovanni, F.; Indurkha, B.; Venture, G. Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review. *J. Comput. Lang.* **2020**, *58*, 100970. [\[CrossRef\]](#)
27. Ziafati, P.; Lera, F.; Costa, A.; Nazarikhorram, A.; Van Der Torre, L.; Nazarikhor, A. ProCrob architecture for personalized social robotics. In Proceedings of the Robots for Learning Workshop@ HRI, Vienna, Austria, 6–9 March 2017; pp. 6–9.
28. Mateo, C.; Brunete, A.; Gambao, E.; Hernando, M. Hammer: An Android based application for end-user industrial robot programming. In Proceedings of the 2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA), Senigallia, Italy, 10–12 September 2014; pp. 1–6. [\[CrossRef\]](#)
29. Datta, C.; MacDonald, B.A. Architecture of an extensible visual programming environment for authoring behaviour of personal service robots. In Proceedings of the 2017 First IEEE International Conference on Robotic Computing (IRC), Taichung, Taiwan, April 2017; pp. 156–159. [\[CrossRef\]](#)
30. Ajaykumar, G.; Steele, M.; Huang, C.M. A survey on end-user robot programming. *ACM Comput. Surv. CSUR* **2021**, *54*, 1–36. [\[CrossRef\]](#)

31. Ur, B.; McManus, E.; Pak Yong Ho, M.; Littman, M.L. Practical trigger-action programming in the smart home. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Toronto, ON, Canada, 26 April–1 May 2014; pp. 803–812. [\[CrossRef\]](#)
32. Huang, J.; Cakmak, M. Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In Proceedings of the 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Vienna, Austria, 6–9 March 2017; pp. 453–462.
33. Pasternak, E.; Fenichel, R.; Marshall, A.N. Tips for creating a block language with blockly. In Proceedings of the 2017 IEEE Blocks and Beyond Workshop (B&B), Raleigh, NC, USA, 9–10 October 2017; pp. 21–24. [\[CrossRef\]](#)
34. Paxton, C.; Hundt, A.; Jonathan, F.; Guerin, K.; Hager, G.D. CoSTAR: Instructing collaborative robots with behavior trees and vision. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May 2017–3 June 2017; pp. 564–571.
35. Steinmetz, F.; Wollschläger, A.; Weitschat, R. Razer—A hri for visual task-level programming and intuitive skill parameterization. *IEEE Robot. Autom. Lett.* **2018**, *3*, 1362–1369. [\[CrossRef\]](#)
36. Connolly, C. Technology and applications of ABB RobotStudio. *Ind. Robot. Int. J.* **2009**, *36*, 540–545. [\[CrossRef\]](#)
37. Georgi, W.; Hohl, P. *Einführung in LabVIEW*; Carl Hanser Verlag GmbH Co KG: Munich, Germany, 2015.
38. Weichhart, G.; Reiser, M.; Stary, C. Task-Based Design of Cyber-Physical Systems—Meeting Representational Requirements with S-BPM. In Proceedings of the International Conference on Subject-Oriented Business Process Management, Bremen, Germany, 2–3 December 2020; pp. 63–73.
39. Bradshaw, J.M. An introduction to software agents. *Softw. Agents* **1997**, *4*, 3–46.
40. Wooldridge, M. Intelligent agents. *Multiagent Syst. Mod. Approach Distrib. Artif. Intell.* **1999**, *1*, 27–73.
41. Iñigo-Blasco, P.; Diaz-del Rio, F.; Romero-Ternero, M.C.; Cagigas-Muñiz, D.; Vicente-Diaz, S. Robotics software frameworks for multi-agent robotic systems development. *Robot. Auton. Syst.* **2012**, *60*, 803–821. [\[CrossRef\]](#)
42. Herrero, H.; Abou Moughlbay, A.; Outón, J.L.; Sallé, D.; de Ipiña, K.L. Skill based robot programming: Assembly, vision and Workspace Monitoring skill interaction. *Neurocomputing* **2017**, *255*, 61–70. [\[CrossRef\]](#)
43. Akkaladevi, S.C.; Pichler, A.; Plach, M.; Ikeda, M.; Hofmann, M. Skill-based programming of complex robotic assembly tasks for industrial application. *e & i Elektrotechnik Informationstechnik* **2019**, *136*, 326–333.
44. Spitzer, F.; Lindorfer, R.; Froschauer, R.; Hofmann, M.; Ikeda, M. A generic Approach for the Industrial Application of Skill-based Engineering using OPC UA. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1671–1678. [\[CrossRef\]](#)
45. Saukkoriipi, J.; Heikkilä, T.; Ahola, J.M.; Seppälä, T.; Isto, P. Programming and control for skill-based robots. *Open Eng.* **2020**, *10*, 368–376. [\[CrossRef\]](#)
46. Giberti, H.; Abbattista, T.; Carnevale, M.; Giagu, L.; Cristini, F. A Methodology for Flexible Implementation of Collaborative Robots in Smart Manufacturing Systems. *Robotics* **2022**, *11*, 9. [\[CrossRef\]](#)
47. Pedersen, M.R.; Nalpantidis, L.; Andersen, R.S.; Schou, C.; Bøgh, S.; Krüger, V.; Madsen, O. Robot skills for manufacturing: From concept to industrial deployment. *Robot. Comput. Integr. Manuf.* **2016**, *37*, 282–291. [\[CrossRef\]](#)
48. Fikes, R.E.; Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* **1971**, *2*, 189–208. [\[CrossRef\]](#)
49. Bøgh, S.; Nielsen, O.S.; Pedersen, M.R.; Krüger, V.; Madsen, O. Does your robot have skills? In Proceedings of the 43rd International Symposium on Robotics, Taipei, Taiwan, 29–31 August 2012; VDE Verlag GMBH: Offenbach, Germany, 2012.
50. Schou, C.; Andersen, R.S.; Chrysostomou, D.; Bøgh, S.; Madsen, O. Skill-based instruction of collaborative robots in industrial settings. *Robot. Comput. Integr. Manuf.* **2018**, *53*, 72–80. [\[CrossRef\]](#)
51. Innerbichler, J.; Gonul, S.; Damjanovic-Behrendt, V.; Mandler, B.; Strohmeier, F. NIMBLE collaborative platform: Microservice architectural approach to federated IoT. In Proceedings of the 2017 Global Internet of Things Summit (GloTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–6.
52. Radic, M.; Vosen, A. Ethische, rechtliche und soziale Anforderungen an Assistenzroboter in der Pflege. *Z. Gerontol. Geriatr.* **2020**, *53*, 630–636. [\[CrossRef\]](#)
53. Chang, W.L.; Šabanović, S. Interaction expands function: Social shaping of the therapeutic robot PARO in a nursing home. In Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction, Portland, OR USA, 2–5 March 2015; pp. 343–350.
54. Graf, B.; Reiser, U.; Hägele, M.; Mauz, K.; Klein, P. Robotic home assistant Care-O-bot® 3-product vision and innovation platform. In Proceedings of the 2009 IEEE Workshop on Advanced Robotics and its Social Impacts, Tokyo, Japan, 23–25 November 2009; pp. 139–144.
55. Mišeikis, J.; Caroni, P.; Duchamp, P.; Gasser, A.; Marko, R.; Mišeikienė, N.; Zwilling, F.; De Castelbajac, C.; Eicher, L.; Früh, M.; et al. Lio-a personal robot assistant for human-robot interaction and care applications. *IEEE Robot. Autom. Lett.* **2020**, *5*, 5339–5346. [\[CrossRef\]](#)
56. Chui, M.; Manyika, J.; Miremadi, M. Where Machines Could Replace Humans-and Where They Can't (Yet). *McKinsey Q.* Available online: <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/where-machines-could-replace-humans-and-where-they-cant-yet> (accessed on 8 May 2023).
57. Meleis, A.I. *Theoretical Nursing: Development and Progress*; Lippincott Williams & Wilkins: Philadelphia, PA, USA, 2011.

58. Alfaro-LeFevre, R. *Applying Nursing Process: The Foundation for Clinical Reasoning*; Lippincott Williams & Wilkins: Philadelphia, PA, USA, 2012.
59. Pichler, A.; Akkaladevi, S.C.; Ikeda, M.; Hofmann, M.; Plasch, M.; Wögerer, C.; Fritz, G. Towards shared autonomy for robotic tasks in manufacturing. *Procedia Manuf.* **2017**, *11*, 72–82. [[CrossRef](#)]
60. Mackinnon, T.; Freeman, S.; Craig, P. Endo-testing: Unit testing with mock objects. *Extrem. Program. Examined* **2000**, 287–301. Available online: <https://www2.ccs.neu.edu/research/demeter/related-work/extreme-programming/MockObjectsFinal.PDF> (accessed on 8 May 2023).
61. Zörrer, H.; Propst, M.; Schuster, D. Example Video of a Simulation for a Part Delivery Scenario of the Use Case Industry in the Research Project ROBxTASK. Available online: <https://doi.org/10.6084/m9.figshare.21154861.v3> (accessed on 8 May 2023).
62. Carlson, J. *Redis in Action*; Simon and Schuster: Shelter Island, NY, USA, 28 June 2013.
63. Faulkner, L. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behav. Res. Methods Instrum. Comput.* **2003**, *35*, 379–383. [[CrossRef](#)]
64. Lewis, C. *Using the “Thinking-Aloud” Method in Cognitive Interface Design*; IBM TJ Watson Research Center Yorktown Heights: Yorktown Heights, NY, USA, 1982.
65. Zörrer, H.; Schuster, D.; Widmoser, F. Example Video of a Testing Environment for the Healthcare Use Case in the Research Project ROBxTASK. Available online: <https://doi.org/10.6084/m9.figshare.22006601.v3> (accessed on 8 May 2023).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.